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-22

Студент: Филатов А. К.

Преподаватель: Миронов Е.С.

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

Дата: 09.12.23

Москва, 2023

**Постановка задачи**

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. 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 1.**

Отсортировать массив целых чисел при помощи битонической сортировки.

**Общий метод и алгоритм решения**

Параметром запуска программы мы указываем максимальное количество используемых потоков. Далее мы указываем длину массива, и создаем массив дополнив длину до ближайшего числа 2^n. Так как это необходимо для работы алгоритма сортировки. Далее считываем данные, а оставшийся участок массива заполняем максимально возможными элементами, чтобы после сортировки они оказались в конце массива, и мы могли также легко их удалить.

Далее вызывается функция сортировки. Сначала мы рекурсивно разбиваем массив пополам, задавая целевое направление сначала вниз, а потом вверх, чтобы получить битонную последовательность. При этом пока у нас есть свободные потоки мы выделяем под вторую половину отдельный поток, а когда они закончатся начинаем работать в однопоточном режиме на выделенном участке.

Когда мы достигаем массива из одного элемента его можно считать отсортированным, как по возрастанию, так и по убыванию. Поэтому два соседних элемента можно считать битонной последовательностью, которую можно собрать в одну возрастающую или убывающую.

Начинаем объединять битонные последовательности. Находим расстояние между двумя элементами, которые будем сравнивать. Оно равно половине участка массива. Далее мы проходим по половине участка и сравниваем каждый элемент с элементом через заданное расстояние. При необходимости меняем их местами. Далее разбиваем данный участок на 2 и повторяем слияние, и так пока его длина не станет равна 1. Таким образом мы получаем битоническую последовательность большего размера, к которой можно опять применить слияние. Повторяем эту процедуру до окончания сортировки. При этом, когда мы будем делать слияние для 2 участков, созданных разными потоками, слияние мы опять разбиваем на несколько потоков, пока это возможно. При всем этом если один поток подготовил последовательность, а второй еще нет, то первый его ждет.

**Код программы**

**bitonic.h**

#pragma once

#include "pthread.h"

#define UP 1

#define DOWN 0

typedef struct ArgsBitonic{

    int \*array;

    int size;

    int start;

    int dir;

}ArgsBitonic;

void InitArgs(ArgsBitonic \*args, int \*array, int size, int start, int dir);

void Comparator(int \*array, int i, int j, int dir);

void BitonicMergeSingleThread(ArgsBitonic \*args);

void BitonicSortSingleThread(ArgsBitonic \*args);

void BitonicMergeMultiThreads(ArgsBitonic \*args);

void BitonicSortMultiThreads(ArgsBitonic \*args);

void bitonicsort(int \*array, int size, int threads);

**bitonic.c**

#include "pthread.h"

#include "bitonic.h"

#include "stdio.h"

#define UP 1

#define DOWN 0

pthread\_mutex\_t lock;

size\_t max\_threads = 1;

size\_t use\_threads = 1;

void InitArgs(ArgsBitonic \*args, int \*array, int size, int start, int dir){

    args->array = array;

    args->size  = size;

    args->start = start;

    args->dir   = dir;

}

void Comparator(int \*array, int i, int j, int dir){

    if(dir == (array[i] > array[j])){

        int temp = array[i];

        array[i] = array[j];

        array[j] = temp;

    }

}

void BitonicMergeSingleThread(ArgsBitonic \*args){

    if(args->size > 1){

        int nextsize = args->size / 2;

        for(int i = args->start; i < nextsize + args->start; ++i){

                Comparator(args->array, i, i + nextsize, args->dir);    // сравнение и перестановка элементов

        }

        ArgsBitonic args1;

        ArgsBitonic args2;

        InitArgs(&args1, args->array, nextsize, args->start, args->dir);

        InitArgs(&args2, args->array, nextsize, args->start + nextsize, args->dir);

        BitonicMergeSingleThread(&args1);

        BitonicMergeSingleThread(&args2);

    }

}

void BitonicSortSingleThread(ArgsBitonic \*args){

    if(args->size > 1){

        int nextsize = args->size / 2;

        ArgsBitonic args1;

        ArgsBitonic args2;

        InitArgs(&args1, args->array, nextsize, args->start, DOWN);

        InitArgs(&args2, args->array, nextsize, args->start + nextsize, UP);

        BitonicSortSingleThread(&args1);

        BitonicSortSingleThread(&args2);

        BitonicMergeSingleThread(args);

    }

}

void BitonicMergeMultiThreads(ArgsBitonic \*args){

    if(args->size > 1){

        int nextsize = args->size / 2;

        int isParal = 0;

        pthread\_t tid;

        for(int i = args->start; i < nextsize + args->start; ++i){      // сравнение и перестановка элементов

                Comparator(args->array, i, i + nextsize, args->dir);

        }

        ArgsBitonic args1;

        ArgsBitonic args2;

        InitArgs(&args1, args->array, nextsize, args->start, args->dir);

        InitArgs(&args2, args->array, nextsize, args->start + nextsize, args->dir);

        pthread\_mutex\_lock(&lock);

        if(use\_threads < max\_threads){

            ++use\_threads;

            pthread\_mutex\_unlock(&lock);

            isParal = 1;

            pthread\_create(&tid, NULL,(void\*) &BitonicMergeMultiThreads, &args1);

            BitonicMergeMultiThreads(&args2);

        } else {

            pthread\_mutex\_unlock(&lock);

            BitonicMergeSingleThread(&args1);

            BitonicMergeSingleThread(&args2);

        }

        if(isParal){

            pthread\_join(tid, NULL);

            pthread\_mutex\_lock(&lock);

            --use\_threads;

            pthread\_mutex\_unlock(&lock);

        }

    }

}

void BitonicSortMultiThreads(ArgsBitonic \*args){

    if(args->size > 1 ){    // Проверяется, если размер массива больше 1, иначе сортировка не требуется.

        int nextsize = args->size / 2;  // Вычисляется размер следующей половины массива nextsize на основе текущего размера args->size.

        int isParal = 0;

        pthread\_t tid;

        ArgsBitonic args1;

        ArgsBitonic args2;

        InitArgs(&args1, args->array, nextsize, args->start, DOWN);

        InitArgs(&args2, args->array, nextsize, args->start + nextsize, UP);

        pthread\_mutex\_lock(&lock);

        if(use\_threads < max\_threads){

            ++use\_threads;

            pthread\_mutex\_unlock(&lock);

            isParal = 1;

            pthread\_create(&tid, NULL,(void\*) &BitonicSortMultiThreads, &args1);

            BitonicSortMultiThreads(&args2);

        } else {

            pthread\_mutex\_unlock(&lock);

            BitonicSortSingleThread(&args1);

            BitonicSortSingleThread(&args2);

        }

        if(isParal){

            pthread\_join(tid, NULL);    // Функция pthread\_join() блокирует вызывающий поток, пока указанный поток не завершится.

            pthread\_mutex\_lock(&lock);

            --use\_threads;

            pthread\_mutex\_unlock(&lock);

        }

        BitonicMergeMultiThreads(args);     // объединение отсортированных половин массива args.

    }

}

void bitonicsort(int \*array, int size, int threads){

    pthread\_mutex\_init(&lock, NULL);    // Инициализируем мьютекс с использованием стандартных атрибутов

    ArgsBitonic args;

    InitArgs(&args,array,size,0,UP);

    if(threads > 1)

        max\_threads = threads;

    BitonicSortMultiThreads(&args);

    pthread\_mutex\_destroy(&lock);

}

**main.c**

#include "stdio.h"

#include "stdlib.h"

#include "bitonic.h"

#include "sys/time.h"

#define MAXINT 2147483647

int SizeStep(int Num){     // Функия нахождения ближайшего числа i, являющегося степенью двойки, для заполнения массива

    int i = 1;

    while(i < Num)

        i \*= 2;

    return i;

}

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

    int threads = 1;    // По умолчанию 1 поток

    if(argc == 2){      // Если пользователь ввёл количество потоков

        threads = atoi(argv[1]);

    }

    int input\_size;     //Вводим количество элементов массива

    scanf("%d",&input\_size);

    //находим ближашее число 2^k >= input\_size

    int size\_array = SizeStep(input\_size);

    int \*array = malloc(sizeof(int)\*size\_array);

    for(int i = 0; i < input\_size; ++i)     // Заполняем массив элементами и, в случае необходимости, дополняем "заглушками"

        scanf("%d",array+i);

    for(int i = input\_size; i < size\_array; ++i)

        array[i] = MAXINT;

    struct timeval start, end;

    gettimeofday(&start, NULL); // получаем время начала выполнения программы

    bitonicsort(array, size\_array, threads);

    gettimeofday(&end, NULL); // получаем время окончания выполнения программы

    bitonicsort(array, size\_array, threads);    // Сортируем массив

    for(int i=0;i<input\_size;++i){      // Выводим результат

        printf("%d\n",array[i]);

    }

    free(array);    //Освобождаем память

    double elapsed\_time = (end.tv\_sec - start.tv\_sec) + ((end.tv\_usec - start.tv\_usec) / 1000000.0);

    printf("Elapsed time: %.6lf seconds\n", elapsed\_time); // выводим время выполнения программы

    return 0;

}

**Протокол работы программы**

**Тестирование:**

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ make clean

rm -r \*.o main

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ make

gcc -c -Wall main.c

gcc -c -Wall bitonic.c

gcc main.o bitonic.o -pthread -o main

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ ./main 1

8

8 7 6 5 4 3 2 1

1

2

3

4

5

6

7

8

Elapsed time: 0.000002 seconds

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ ./main 2

13

3 1 4 2 5 8 9 6 7 10 13 23 22

1

2

3

4

5

6

7

8

9

10

13

22

23

Elapsed time: 0.000673 seconds

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ ./main 4

10

10 9 8 7 6 5 4 3 2 1

1

2

3

4

5

6

7

8

9

10

Elapsed time: 0.001379 seconds

// Время выполнения для больших файлов (10 тыс. элементов) с разным количеством потоков:

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ cat test.txt | ./main 1

Elapsed time: 0.006400 seconds

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ cat test.txt | ./main 2

Elapsed time: 0.003640 seconds

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ cat test.txt | ./main 3

Elapsed time: 0.003503 seconds

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ cat test.txt | ./main 4

Elapsed time: 0.002677 seconds

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ cat test.txt | ./main 5

Elapsed time: 0.002371 seconds

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ cat test.txt | ./main 6

Elapsed time: 0.002276 seconds

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ cat test.txt | ./main 7

Elapsed time: 0.002680 seconds

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ cat test.txt | ./main 8

Elapsed time: 0.003241 seconds

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во потоков | Время (сек) | Ускорение | Эффективность |
| 1 | 0.006400 | 1 | 1 |
| 2 | 0.003640 | 1.76 | 0.88 |
| 4 | 0.002677 | 2.39 | 0.60 |
| 8 | 0.003241 | 1.97 | 0.25 |

**Strace:**

cblphblu@DESKTOP-3PDEL6G:/mnt/c/Users/user/Desktop/МАИ/2 курс/ОСИ/LR2FILATOV$ strace -f ./main 2

execve("./main", ["./main", "2"], 0x7ffe746b9af0 /\* 19 vars \*/) = 0

brk(NULL) = 0x55da0b751000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fff77337dd0) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f2c0e874000

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=34303, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 34303, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f2c0e86b000

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\0i8\235HZ\227\223\333\350s\360\352,\223\340."..., 68, 896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0644, 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) = 0x7f2c0e643000

mmap(0x7f2c0e66b000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f2c0e66b000

mmap(0x7f2c0e800000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f2c0e800000

mmap(0x7f2c0e858000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x214000) = 0x7f2c0e858000

mmap(0x7f2c0e85e000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f2c0e85e000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f2c0e640000

arch\_prctl(ARCH\_SET\_FS, 0x7f2c0e640740) = 0

set\_tid\_address(0x7f2c0e640a10) = 1350

set\_robust\_list(0x7f2c0e640a20, 24) = 0

rseq(0x7f2c0e6410e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f2c0e858000, 16384, PROT\_READ) = 0

mprotect(0x55da0a304000, 4096, PROT\_READ) = 0

mprotect(0x7f2c0e8ae000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7f2c0e86b000, 34303) = 0

newfstatat(0, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

getrandom("\x27\x10\x6e\x64\x90\x4e\xbe\xb4", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55da0b751000

brk(0x55da0b772000) = 0x55da0b772000

read(0, 8

"8\n", 1024) = 2

read(0, 8 7 6 5 4 3 2 1

"8 7 6 5 4 3 2 1\n", 1024) = 16

rt\_sigaction(SIGRT\_1, {sa\_handler=0x7f2c0e6d48f0, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_ONSTACK|SA\_RESTART|SA\_SIGINFO, sa\_restorer=0x7f2c0e685520}, 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) = 0x7f2c0de3f000

mprotect(0x7f2c0de40000, 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=0x7f2c0e63f910, parent\_tid=0x7f2c0e63f910, exit\_signal=0, stack=0x7f2c0de3f000, stack\_size=0x7fff00, tls=0x7f2c0e63f640}strace: Process 1351 attached**

**=> {parent\_tid=[1351]}, 88) = 1351**

[pid 1351] rseq(0x7f2c0e63ffe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 1350] rt\_sigprocmask(SIG\_SETMASK, [], <unfinished ...>

[pid 1351] <... rseq resumed>) = 0

[pid 1351] set\_robust\_list(0x7f2c0e63f920, 24 <unfinished ...>

[pid 1350] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 1351] <... set\_robust\_list resumed>) = 0

[pid 1350] futex(0x7f2c0e63f910, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 1351, NULL, FUTEX\_BITSET\_MATCH\_ANY <unfinished ...>

[pid 1351] rt\_sigprocmask(SIG\_SETMASK, [], NULL, 8) = 0

[pid 1351] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], NULL, 8) = 0

[pid 1351] madvise(0x7f2c0de3f000, 8368128, MADV\_DONTNEED) = 0

[pid 1351] exit(0) = ?

[pid 1350] <... futex resumed>) = 0

[pid 1351] +++ exited with 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=0x7f2c0e63f910, parent\_tid=0x7f2c0e63f910, exit\_signal=0, stack=0x7f2c0de3f000, stack\_size=0x7fff00, tls=0x7f2c0e63f640}strace: Process 1352 attached**

**=> {parent\_tid=[1352]}, 88) = 1352**

[pid 1352] rseq(0x7f2c0e63ffe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 1350] rt\_sigprocmask(SIG\_SETMASK, [], <unfinished ...>

[pid 1352] <... rseq resumed>) = 0

[pid 1350] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 1352] set\_robust\_list(0x7f2c0e63f920, 24 <unfinished ...>

[pid 1350] futex(0x7f2c0e63f910, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 1352, NULL, FUTEX\_BITSET\_MATCH\_ANY <unfinished ...>

[pid 1352] <... set\_robust\_list resumed>) = 0

[pid 1352] rt\_sigprocmask(SIG\_SETMASK, [], NULL, 8) = 0

[pid 1352] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], NULL, 8) = 0

[pid 1352] madvise(0x7f2c0de3f000, 8368128, MADV\_DONTNEED) = 0

[pid 1352] exit(0) = ?

[pid 1350] <... futex resumed>) = 0

[pid 1352] +++ exited with 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=0x7f2c0e63f910, parent\_tid=0x7f2c0e63f910, exit\_signal=0, stack=0x7f2c0de3f000, stack\_size=0x7fff00, tls=0x7f2c0e63f640}strace: Process 1353 attached**

**=> {parent\_tid=[1353]}, 88) = 1353**

[pid 1353] rseq(0x7f2c0e63ffe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 1350] rt\_sigprocmask(SIG\_SETMASK, [], <unfinished ...>

[pid 1353] <... rseq resumed>) = 0

[pid 1350] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 1353] set\_robust\_list(0x7f2c0e63f920, 24 <unfinished ...>

[pid 1350] futex(0x7f2c0e63f910, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 1353, NULL, FUTEX\_BITSET\_MATCH\_ANY <unfinished ...>

[pid 1353] <... set\_robust\_list resumed>) = 0

[pid 1353] rt\_sigprocmask(SIG\_SETMASK, [], NULL, 8) = 0

[pid 1353] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], NULL, 8) = 0

[pid 1353] madvise(0x7f2c0de3f000, 8368128, MADV\_DONTNEED) = 0

[pid 1353] exit(0) = ?

[pid 1350] <... futex resumed>) = 0

[pid 1353] +++ exited with 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=0x7f2c0e63f910, parent\_tid=0x7f2c0e63f910, exit\_signal=0, stack=0x7f2c0de3f000, stack\_size=0x7fff00, tls=0x7f2c0e63f640}strace: Process 1354 attached**

**=> {parent\_tid=[1354]}, 88) = 1354**

[pid 1354] rseq(0x7f2c0e63ffe0, 0x20, 0, 0x53053053 <unfinished ...>

[pid 1350] rt\_sigprocmask(SIG\_SETMASK, [], <unfinished ...>

[pid 1354] <... rseq resumed>) = 0

[pid 1350] <... rt\_sigprocmask resumed>NULL, 8) = 0

[pid 1354] set\_robust\_list(0x7f2c0e63f920, 24 <unfinished ...>

[pid 1350] futex(0x7f2c0e63f910, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 1354, NULL, FUTEX\_BITSET\_MATCH\_ANY <unfinished ...>

[pid 1354] <... set\_robust\_list resumed>) = 0

[pid 1354] rt\_sigprocmask(SIG\_SETMASK, [], NULL, 8) = 0

[pid 1354] rt\_sigprocmask(SIG\_BLOCK, ~[RT\_1], NULL, 8) = 0

[pid 1354] madvise(0x7f2c0de3f000, 8368128, MADV\_DONTNEED) = 0

[pid 1354] exit(0) = ?

[pid 1350] <... futex resumed>) = 0

[pid 1354] +++ exited with 0 +++

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

write(1, "1\n", 21

) = 2

write(1, "2\n", 22

) = 2

write(1, "3\n", 23

) = 2

write(1, "4\n", 24

) = 2

write(1, "5\n", 25

) = 2

write(1, "6\n", 26

) = 2

write(1, "7\n", 27

) = 2

write(1, "8\n", 28

) = 2

write(1, "Elapsed time: 0.004351 seconds\n", 31Elapsed time: 0.004351 seconds

) = 31

lseek(0, -1, SEEK\_CUR) = -1 ESPIPE (Недопустимая операция смещения)

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

Многие языки программирования позволяют пользователю работать с потоками. Создание потоков происходит быстрее, чем создание процессов, за счет того, что при создании потока не копируется область памяти, а они все работают с одной областью памяти. Поэтому многопоточность используют для ускорения не зависящих друг от друга, однотипных задач, которые будут работать параллельно.

В данной лабораторной работе я реализовал и исследован алгоритм битонной сортировки. Установив при этом, что, используя 4 потока можно получить выигрыш по времени в 2,5раза. При дальнейшем увеличении потоков прирост почти не увеличивается, а даже может уменьшаться из-за того, что на управление и переключение потоков уходит больше времени, чем они выигрывают. Однако, чем больше входные данные, тем лучше себя показывают большее количество потоков.