Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Управление потоками и синхронизация**

Группа: М80-207Б-20

Студент: Куценко Б.Д.

Преподаватель: Миронов Е.С.

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

Дата:

Москва, 2021

# Постановка задачи

Составить и отладить программу на языке С++, обрабатывающую данные в многопоточном режиме. Использовать стандартные средства создания потоков операционной системы Unix.

**Вариант задания:** 1.

Битонная сортировка.

# Общие сведения о программе

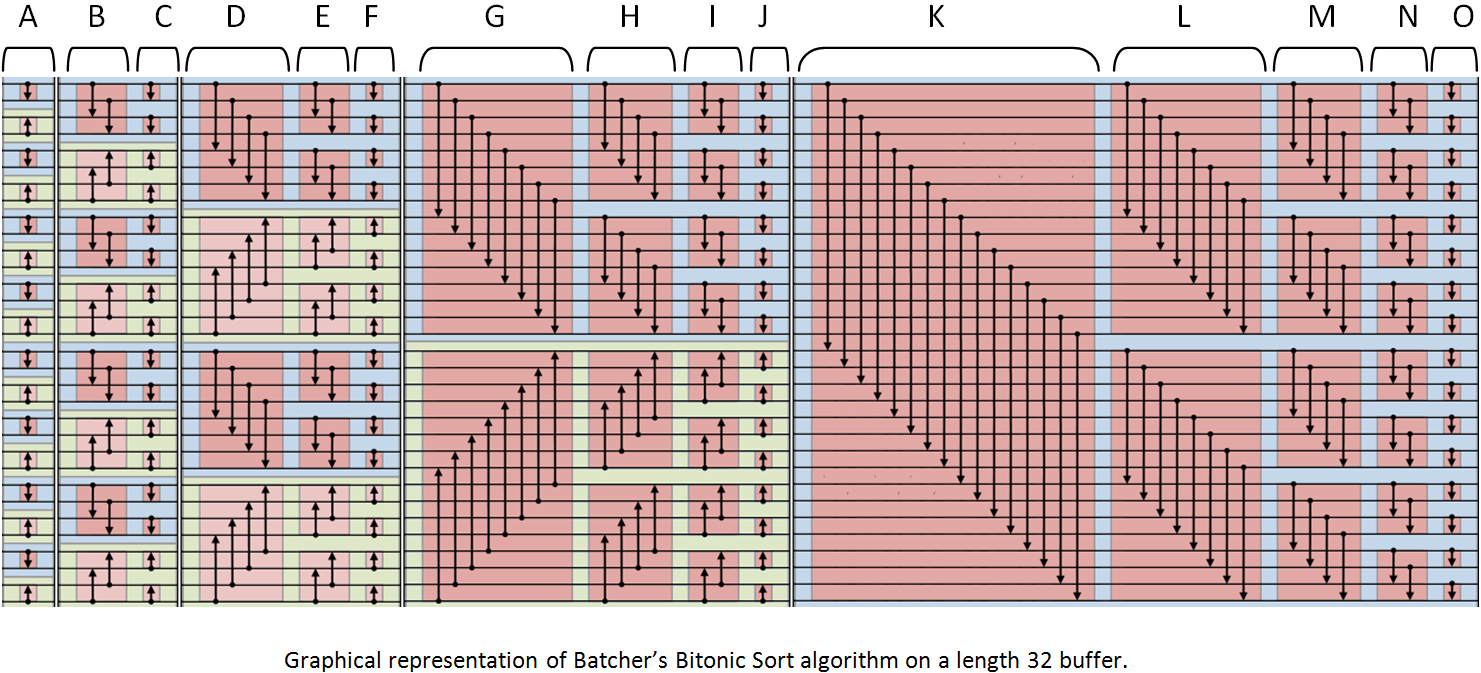
Программа состоит из файла **bitonicsort.c**, в котором реализован многопоточный алгоритм битонной сортировки.

Используются следующие системные вызовы

1. **thread() –** создает новый поток выполнения в программе.
2. **thread.join()** – для синхронного окончания выаполнения функций в потоках

# Общий метод и алгоритм решения

Для битонной сортировки используются сортирующие сети



Они устроены так, что за каждый шаг можно сделать несколько перестановок одновременно.

Сложность такой сортировки равна log^2n при количестве потоков n/2.

**Код программы**

**main.cpp:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <pthread.h>

#include <unistd.h>

#define MAX 0x7f

#define ASCENDING 1

#define DESCENDING 0

int activeThreads;

int maxThreads = 6;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int for\_test;

typedef struct \_stat {

    int\* arr;

    int start;

    int count;

    int direction;

} stat;

int\* createArray(int n, int\* real);

void print(int \*arr, int amount);

void BitonicSort(int \*arr, int n);

void bitonicSort(int \*arr, int start, int count, int direction);

void\* bitonicSortThread(void \*arg);

void bitonicMerge(int \*arr, int start, int count, int direction);

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

    if(argc == 2) {

        maxThreads = atoi(argv[1]);

    }

    int i;

    int n, real;

    printf("Enter amount of numbers\n");

    scanf("%d", &n);

    int\* arr = createArray(n, &real);

    for\_test = real;

    for (i = 0; i < n; i++) {

        arr[i] = rand() % n; // (N - i);

    }

    printf("n = %d, real = %d\n", n, real);

    // for (int i = 0; i < n; i++) {

    //     scanf("%d", arr + i);

    // }

    printf("Not sorted array\n");

    // print(arr, real);

    BitonicSort(arr, real);

    printf("Sorted array\n");

    print(arr, n);

}

int\* createArray(int n, int\* real) {

    \*real = (n > 0) ? 1 : 0;

    while(n > \*real) {

        \*real \*= 2;

    }

    int \*arr = malloc(\*real \* sizeof(int));

    memset(arr, MAX, \*real \* sizeof(int));

    return arr;

}

void BitonicSort(int \*arr, int n) {

    activeThreads = 1;

    bitonicSort(arr, 0, n, ASCENDING);

}

void\* bitonicSortThread(void \*arg) {

    if(((stat \*)arg)->count > 1)

        bitonicSort(((stat \*)arg)->arr, ((stat \*)arg)->start, ((stat \*)arg)->count, ((stat \*)arg)->direction);

}

void bitonicSort(int \*arr, int start, int count, int direction) {

    if (count < 2) return;

    int newCount;

    newCount = count / 2;

    pthread\_t thread1;

    stat stat1;

    stat1.arr = arr;

    stat1.start = start;

    stat1.count = newCount;

    stat1.direction = ASCENDING;

    pthread\_t thread2;

    stat stat2;

    stat2.arr = arr;

    stat2.start = start + newCount;

    stat2.count = newCount;

    stat2.direction = DESCENDING;

        // pthread\_mutex\_lock(&mutex);

        // activeThreads++;

        // activeThreads++;

        // pthread\_mutex\_unlock(&mutex);

    pthread\_create(&thread1, NULL, bitonicSortThread, &stat1);

    pthread\_create(&thread2, NULL, bitonicSortThread, &stat2);

    pthread\_join(thread1, NULL);

    pthread\_join(thread2, NULL);

    pthread\_mutex\_lock(&mutex);

        // activeThreads--;

        // activeThreads--;

    pthread\_mutex\_unlock(&mutex);

    pthread\_mutex\_lock(&mutex);

    bitonicMerge(arr, start, count, direction);

    pthread\_mutex\_unlock(&mutex);

    // } else {

        // pthread\_mutex\_unlock(&mutex);

        // sleep(1);

        // bitonicSort(arr, start, count, direction);

    // }

}

void swap(int\* arr, int i, int j, int direction)

{

    int tmp;

    if (direction==(arr[i]>arr[j])) {

        tmp = arr[i];

        arr[i] = arr[j];

        arr[j] = tmp;

    }

}

void bitonicMerge(int \*arr, int start, int count, int direction)

{

    int newCount, i;

    if (count > 1) {

        newCount = count/2;

        for (i = start; i < start+newCount; i++)

            swap(arr, i, i+newCount, direction);

        bitonicMerge(arr, start, newCount, direction);

        bitonicMerge(arr, start+newCount, newCount, direction);

    }

}

void print(int \*arr, int amount) {

    putchar('[');

    for (int i = 0; i < amount; i ++) {

        (i == amount - 1) ? printf("%d", arr[i]) : printf("%d,", arr[i]);

    }

    putchar(']');

    putchar('\n');

}

# Демонстрация работы программы в терминале

**Пример 1:**

Enter amount of numbers

10

n = 10, real = 16

Not sorted array

[3,6,7,5,3,5,6,2,9,1]

Sorted array

[1,2,3,3,5,5,6,6,7,9]

**Пример 2:**

Enter amount of numbers

20

n = 20, real = 32

Not sorted array

[3,6,17,15,13,15,6,12,9,1,2,7,10,19,3,6,0,6,12,16]

Sorted array

[0,1,2,3,3,6,6,6,6,7,9,10,12,12,13,15,15,16,17,19]

# Сравнение эффективности работы:

Так как этот алгоритм для видеокарт, то приходится создавать очень много процессов, что сильно нагружает систему, поэтому он не так эффективен на больших данных, как хотелось бы.

# Вывод

В данной ЛР я познакомился с многопоточностью и синхронизацией потоков с помощью мьютексов. Я выбрал алгоритм для сортирующих сетей, который используется в основном для видеокарт, где можно делать много одинаковых действий. К сожалению, количество потоков очень быстро растет, что очень сильно загружает систему и тормозит время отклика системных вызовов, что делает этот алгоритм неэффективным для больших данных, используя ресурсы процессора.