Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Студент: Стрыгин Д.Д.

Группа: М8О–306Б–19

Вариант: 4

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Постановка задачи**

**Цель работы**

Целью является приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками

**Задание**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков 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 входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

Отсортировать массив целых чисел при помощи сортировки Бэтчера.

**Общие сведения о программе**

Программа компилируется из файлов main.c, batcher\_sort.c. Также используется заголовочные файлы: pthread.h, stdio.h , stdlib.h, sys/time.h. В программе используются следующие системные вызовы:

1. **pthread\_create** – создает поток.
2. **pthread\_join** – блокирует вызывающий поток, пока указанный поток не завершится.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы с потоками и параллельной сортировкой Бэтчера.
2. Написать программу, которая будет работать с заданным количеством потоков и сравнить ускорение и эффективность в зависимости от количества потоков.
3. Написать функции для многопоточной сортировки Бэтчера.

**Основные файлы программы**

**batcher.c:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <limits.h>

#include "sys/time.h"

#include "batcher.h"

// Текущее кол-во потоков

int curthread\_count = 0;

extern int maxthread\_count;

extern int n;

extern pthread\_t\* thread\_handles;

// Функция сортировки Бэтчера распоточевается рекурсивно

void\* batcher\_mergesort() {

   if (r == (l + 1))

      compexch(list, l, r);

   if (r < (l + 2))

      return;

   unsuflle(list, l, r);

   if (curthread\_count != maxthread\_count) {

      if(pthread\_create(&thread\_handles[curthread\_count - 1], NULL, batcher\_mergesort, (void\*)(list, l, (l + r - 1) / 2)) != 0) {

         fprintf(stderr, "PTHREAD CREATE ERROR\n");

         exit(-1);

      }

      else

         curthread\_count++;

      if(pthread\_create(&thread\_handles[curthread\_count], NULL, batcher\_mergesort, (void\*)(list, (l + r + 1) / 2, r)) != 0) {

         fprintf(stderr, "PTHREAD CREATE ERROR\n");

         exit(-1);

      }

      else

         curthread\_count++;

      if(pthread\_join(thread\_handles[curthread\_count - 1], NULL) != 0) {

         fprintf(stderr, "PTHREAD JOIN ERROR\n");

         exit(-1);

      }

      if(pthread\_join(thread\_handles[curthread\_count], NULL) != 0) {

         fprintf(stderr, "PTHREAD JOIN ERROR\n");

         exit(-1);

      }

   }

   else {

      batcher\_mergesort(list, l, (l + r - 1) / 2);

      batcher\_mergesort(list, (l + r + 1) / 2, r);

   }

   suffle(list, l, r);

   for (int i = l + 1; i < r; i += 2)

      compexch(list, i, i + 1);

}

// Вспомогательные ф-ии для сортировки

void shuffle(int list [], int l, int r) {

   int i, j, m = (l + r + 1) / 2;

   int buf[2 \* m];

   for (i = l, j = 0; i <= r; i += 2, j++) {

      buf[i] = list[l + j];

      buf[i + 1] = list[m + j];

   }

   for (i = l; i <= r; i++)

      list[i] = buf[i];

}

void unsuflle(int list [], int l, int r) {

   int i, j, m = (l + r + 1) / 2;

   int buf[2 \* m];

   for (i = l, j = 0; i <= r; i += 2, j++) {

      buf[l + j] = list[i];

      buf[m + j] = list[i + 1];

   }

   for (i = l; i <= r; i++)

      list[i] = buf[i];

}

void compexch(int list [], int l, int r) {

   int buf;

   if (list[l] > list[r])

      buf = list[r];

      list[r] = list[l];

      list[l] = buf;

}

**Batcher.h:**

#ifndef \_\_BATCHERMERGESORT\_

#define \_\_BATCHERMERGESORT\_

void\* batcher\_mergesort(void\*);

void shuffle(int, int, int);

void unshuffle(int, int, int);

void compexch(int, int, int);

#endif

**lab3.c**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <limits.h>

#include <time.h>

#include "sys/time.h"

#include "batcher.h"

#include "table.h"

#define MYMAX 1000

//глобальные переменные

int bar\_count = 0;

int n = 0;

int maxthread\_count = 0;

int \*list1 = NULL;

int \*l\_a = NULL;

pthread\_t\* thread\_handles = NULL;    // Массив указателей на поток

//функция вычисления текущего времени

long long current\_timestamp() {

   struct timeval te;

   if(gettimeofday(&te, NULL) != 0) {

      fprintf(stderr, "GETTIME ERROR\n");

      exit(-1);

   }

   long long milliseconds = te.tv\_sec \* 1000LL + te.tv\_usec / 1000;

   return milliseconds;

}

//функция справки использования программы

void usage(char\* prog\_name) {

   fprintf(stderr, "usage: %s <thread count> <n> [g] [o]\n", prog\_name);

   fprintf(stderr, "'values[0]': кол-во потоков\n");

   fprintf(stderr, "values[0] не должно быть равно 0\n");

   fprintf(stderr, "оперция log2(values[0]) должна возвращать целое число\n");

   fprintf(stderr, "'n': кол-во элементов в массиве\n");

   fprintf(stderr, "n должно делиться на кол-во потоков\n");

   fprintf(stderr, "values[0] не может быть больше, чем n/2\n");

   fprintf(stderr, "'g':  генерирует числа\n");

   fprintf(stderr, "'o':  программа выведет и входные данные\n");

   exit(-1);

}

// Работа с ключами

void get\_args(int argc, char \*argv[], int\* gen\_list\_p, int\* output\_list\_p, int values [], int nvalue) {

   int values[0] = 0, n = 0;

   if (argc < 3 || argc > 5)

      usage(argv[0]);

   //Количество потоков и количество чисел в массиве

   if((values[0] = strtol(argv[1], NULL, 10)) == LONG\_MIN || values[0] == LONG\_MAX) {

      fprintf(stderr, "STRTOL ERROR\n");

      exit(-1);

   }

   if((n = strtol(argv[2], NULL, 10)) == LONG\_MIN || n == LONG\_MAX) {

      fprintf(stderr, "STRTOL ERROR\n");

      exit(-1);

   }

   //Проверка на корректность кол-ва потоков

   if(values[0] == 0)

      usage(argv[0]);

   if (n % values[0] != 0)

      usage(argv[0]);

   if ((values[0] & (values[0]-1)) != 0)

      usage(argv[0]);

   if (values[0] > (n / 2))

         usage(argv[0]);

   // Указатели на массивы ввода-вывода

   \*gen\_list\_p = 0;

   \*output\_list\_p = 0;

   // Проверка корректности 2х последних ключей

   if (argc == 4) {

      char c1 = argv[3][0];

      if (c1 == 'g')

         \*gen\_list\_p = 1;

      if (c1 == 'o')

         \*output\_list\_p = 1;

      else {

         printf(stderr, "Ошибка ввода ключей\n");

         exit(-1);

      }

   }

   else if (argc == 5) {

      char c1 = argv[3][0];

      char c2 = argv[4][0];

      if ((c1 == 'g') && (c2 == 'o')) {

         \*gen\_list\_p = 1;

         \*output\_list\_p = 1;

      }

      else {

         printf(stderr, "Ошибка ввода ключей\n");

         exit(-1);

      }

   }

   values[0] = values[0];

   values[1] = n;

}

//функция генерации массива

void gen\_list(int list[], int n) {

   srand(time(NULL));

   for (int i = 0; i < n; ++i) { list[i] = 1 + rand()%MYMAX; }

}

//функция чтения массива

void read\_list(char prompt[], int list[], int n) {

   printf("%s\n", prompt);

   for (int i = 0; i < n; ++i) { scanf("%d", &list[i]); }

}

//функция печати на экран массива

void print\_list(char title[], int list[], int n) {

   printf("%s:\n", title);

   for (int i = 0; i < n; ++i) { printf("%d\n", list[i]); }

}

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

   int g\_list = 0, output\_list = 0, nvalue = 2;

   long long start = 0, finish = 0;

   int values[2];

   get\_args(argc, argv, &g\_list, &output\_list, values, nvalue);

   if((thread\_handles = malloc(values[0]\*sizeof(pthread\_t))) == NULL) {

      fprintf(stderr, "MALLOC ERROR\n");

      exit(-1);

   }

   if((list1 = malloc(values[1]\*sizeof(int))) == NULL) {

      fprintf(stderr, "MALLOC ERROR\n");

      exit(-1);

   }

   if (g\_list)

      gen\_list(list1, values[1]);

   else

      read\_list("Enter the list", list1, values[1]);

   if (output\_list)

      print\_list("The input list is", list1, values[1]);

   args\_t argss = {list1, 0, n - 1};

   // Начало отсчёта времени

   start = current\_timestamp();

   // Сортировка

   batcher\_mergesort(&argss);

   // Конец отсчёта времени

   finish = current\_timestamp();

   if (output\_list)

      print\_list("The sorted list is", l\_a, values[1]);

   printf("Elapsed time = %lld milliseconds\n", finish - start);

   free(list1);

   free(l\_a);

   free(thread\_handles);

   return 0;

}

**table.h**

#pragma once

typedef struct args {

    int \*list;

    int l, r;

}args\_t;

**Вывод**

На СИ можно писать программы использующие распараллеливание, ускорение и эффективность алгоритма зависит от количества данных и количества потоков. Однако, эффективное ускорение идёт только до определённого числа потоков, и как правило ограничено кол-вом данных и возможностями железа ПК.