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: Пищик Е.С.

Группа: М8О–206Б–19

Вариант: 1

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**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 битонической сортировки.

**Общие сведения о программе**

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

1. **pthread\_mutex\_lock** – блокировка mutex.
2. **pthread\_mutex\_lock** – разблокировка mutex.
3. **pthread\_cond\_broadcast** – пробуждает блокированный поток по переменной состояния.
4. **pthread\_cond\_wait** - используется, чтобы атомарно освободить мьютекс и заставить вызывающий поток блокироваться по переменной состояния. **pthread\_cond\_wait** возвращает 0 - после успешного завершения.
5. **pthread\_mutex\_init** – инициализация mutex.
6. **pthread\_cond\_init** – инициализация переменной состояния.
7. **pthread\_create** – создает поток.
8. **pthread\_join** – блокирует вызывающий поток, пока указанный поток не завершится.
9. **pthread\_mutex\_destroy -** уничтожение mutex.
10. **pthread\_cond\_destroy –** уничтожение переменной состояния.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

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

**Исследование искорения и эффективности от количества потоков.**

Si – ускорение, где i = количество потоков.

Si = T1/Ti

Ei – эффективность, где i = количество потоков.

Ei = Si / i

Ti – время работы алгоритма, где i = количество потоков.

1. **Массив из 1 млн. чисел:**

S2 = 1.93 E2 = 0.97

S4 = 3.09 E4 = 0.77

S8 = 4.60 E8 = 0.58

S16 = 3.82 E16 = 0.24

S32 = 2.87 E32 = 0.09

S64 = 2.50 E64 = 0.04

На 1 млн. чисел лучшее ускорение показал алгоритм на 8 потоках, причем от 2 до 8 потоков ускорение росло, от 8 до 64 потоков ускорение падало, что говорит о том, что не всегда – больше = лучше! Однако лучшую эффективность показал алгоритм на 2 потоках, от 2 до 64 потоков эффективность падала.

2. **Массив из 10 млн. чисел:**

S2 = 1.99 E2 = 0.99

S4 = 3.93 E4 = 0.98

S8 = 5.98 E8 = 0.75

S16 = 5.37 E16 = 0.34

S32 = 5.09 E32 = 0.16

S64 = 4.58 E64 = 0.07

На 10 млн. чисел лучшее ускорение показал алгоритм на 8 потоках, причем от 2 до 8 потоков ускорение росло, от 8 до 64 потоков ускорение падало, однако по сравнению с 1 млн. ускорение везде возросло, но на 8, 16, 32 и 64 наиболее сильно, ускорение на 32 и 64 потоках, опередило ускорение на 4 потоках. Лучшую эффективность показал алгоритм на 2 потоках, но на 10 млн. уже алгоритм на 4 потоках показал почти идентичную эффективность 0.98 ~ 0.99, но ускорение на 4 потоках значительно выше, чем на 2. От 2 до 64 потоков эффективность падала, но была в каждом пункте выше чем на 1 млн. данных.

**Основные файлы программы**

**main.c:**

//gcc -pthread main.c bitonic\_sort.c

//Пищик Е.С. М8О-206Б-19

//Лабораторная работа №3

//Вариант 1

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <limits.h>

#include "sys/time.h"

#include "bitonic\_sort.h"

//константа

#define MYMAX 1000000

//глобальные переменные

int thread\_count = 0;

int bar\_count = 0;

int n = 0;

int \*list1 = NULL;

int \*list2 = NULL;

int \*l\_a = NULL;

int \*l\_b = NULL;

pthread\_mutex\_t bar\_mutex;

pthread\_cond\_t bar\_cond;

//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 описания использования при неправильном кол-ве аргументов

void usage(char\* prog\_name)

{

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

fprintf(stderr, "thread\_count should be nonzero\n");

fprintf(stderr, "thread\_count should be pow of 2\n");

fprintf(stderr, "n = number of elements in list\n");

fprintf(stderr, "n should be evenly divisible by thread count\n");

fprintf(stderr, "'g': program should generate the list\n");

fprintf(stderr, "'o': program should output original and sorted lists\n");

exit(-1);

}

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

{

if (argc < 3 || argc > 5) { usage(argv[0]); }

//Количество потоков и количество чисел в массиве

if((thread\_count = strtol(argv[1], NULL, 10)) == LONG\_MIN || thread\_count == 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);

}

//Если количество эл-тов массива не делится нацело на количество потоков

if(thread\_count == 0) { usage(argv[0]); }

if (n % thread\_count != 0) { usage(argv[0]); }

if ((thread\_count & (thread\_count-1)) != 0) { usage(argv[0]); }

\*gen\_list\_p = 0;

\*output\_list\_p = 0;

if (argc == 4)

{

char c1 = argv[3][0];

if (c1 == 'g') { \*gen\_list\_p = 1; }

else { \*output\_list\_p = 1; }

}

else if (argc == 5)

{

\*gen\_list\_p = 1;

\*output\_list\_p = 1;

}

}

//функция генрации массива

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

{

srandom(1);

for (int i = 0; i < n; ++i) { list[i] = random() % 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[])

{

pthread\_t\* thread\_handles = NULL;

int g\_list = 0;

int output\_list = 0;

long long start = 0;

long long finish = 0;

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

if((thread\_handles = malloc(thread\_count\*sizeof(pthread\_t))) == NULL)

{

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

exit(-1);

}

if(pthread\_mutex\_init(&bar\_mutex, NULL) != 0)

{

fprintf(stderr, "MUTEX INIT ERROR\n");

exit(-1);

}

if(pthread\_cond\_init(&bar\_cond, NULL) != 0)

{

fprintf(stderr, "COND INIT ERROR\n");

exit(-1);

}

if((list1 = malloc(n\*sizeof(int))) == NULL)

{

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

exit(-1);

}

if((list2 = malloc(n\*sizeof(int))) == NULL)

{

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

exit(-1);

}

l\_a = list1;

l\_b = list2;

if (g\_list) { gen\_list(list1, n); }

else { read\_list("Enter the list", list1, n); }

if (output\_list) { print\_list("The input list is", list1, n); }

start = current\_timestamp();

for (long thread = 0; thread < thread\_count; ++thread)

{

if(pthread\_create(&thread\_handles[thread], NULL, bitonic\_sort, (void\*)thread) != 0)

{

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

exit(-1);

}

}

for (long thread = 0; thread < thread\_count; ++thread)

{

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

{

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

exit(-1);

}

}

finish = current\_timestamp();

if (output\_list) { print\_list("The sorted list is", l\_a, n); }

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

free(list1);

free(list2);

if(pthread\_mutex\_destroy(&bar\_mutex) != 0)

{

fprintf(stderr, "MUTEX DESTROY ERROR\n");

exit(-1);

}

if(pthread\_cond\_destroy(&bar\_cond) != 0)

{

fprintf(stderr, "COND DESTROY ERROR\n");

exit(-1);

}

free(thread\_handles);

return 0;

}

**bitonic\_sort.c:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <limits.h>

#include "sys/time.h"

#include "bitonic\_sort.h"

//Глобальные переменные

extern int thread\_count;

extern int bar\_count;

extern pthread\_mutex\_t bar\_mutex;

extern pthread\_cond\_t bar\_cond;

extern int n;

extern int\* list1;

extern int\* list2;

extern int\* l\_a;

extern int\* l\_b;

//Функция сортировки

void\* bitonic\_sort(void\* rank)

{

long tmp = (long)rank;

int my\_rank = (int)tmp;

int local\_n = n/thread\_count;

int my\_first = my\_rank\*local\_n;

unsigned th\_count, and\_bit, dim;

qsort(list1 + my\_first, local\_n, sizeof(int), compare);

barrier();

for (th\_count = 2, and\_bit = 2, dim = 1; th\_count <= thread\_count;

th\_count <<= 1, and\_bit <<= 1, ++dim)

{

if ((my\_rank & and\_bit) == 0)

{ bitonic\_sort\_incr(th\_count, dim, my\_first, local\_n, my\_rank); }

else

{ bitonic\_sort\_decr(th\_count, dim, my\_first, local\_n, my\_rank); }

}

return NULL;

}

void bitonic\_sort\_incr(int th\_count, int dim, int my\_first, int local\_n, int my\_rank)

{

int partner = 0;;

int\* tmp = NULL;

unsigned eor\_bit = 1 << (dim - 1);

for (int stage = 0; stage < dim; ++stage)

{

partner = my\_rank ^ eor\_bit;

if (my\_rank < partner)

{ merge\_split\_lo(my\_rank, my\_first, local\_n, partner); }

else

{ merge\_split\_hi(my\_rank, my\_first, local\_n, partner); }

eor\_bit >>= 1;

barrier();

if (my\_rank == 0)

{

tmp = l\_a;

l\_a = l\_b;

l\_b = tmp;

}

barrier();

}

}

void bitonic\_sort\_decr(int th\_count, int dim, int my\_first, int local\_n, int my\_rank)

{

int partner = 0;

int\* tmp = NULL;

unsigned eor\_bit = 1 << (dim - 1);

for (int stage = 0; stage < dim; ++stage)

{

partner = my\_rank ^ eor\_bit;

if (my\_rank > partner)

{ merge\_split\_lo(my\_rank, my\_first, local\_n, partner); }

else

{ merge\_split\_hi(my\_rank, my\_first, local\_n, partner); }

eor\_bit >>= 1;

barrier();

if (my\_rank == 0)

{

tmp = l\_a;

l\_a = l\_b;

l\_b = tmp;

}

barrier();

}

}

//Функция верхнего слияния

void merge\_split\_lo(int my\_rank, int my\_first, int local\_n, int partner)

{

int ai = my\_first;

int bi = my\_first;

int xi = partner\*local\_n;

for (int i = 0; i < local\_n; ++i)

{

if (l\_a[ai] <= l\_a[xi]) { l\_b[bi++] = l\_a[ai++]; }

else { l\_b[bi++] = l\_a[xi++]; }

}

}

//Функция нижнего слияния

void merge\_split\_hi(int my\_rank, int my\_first, int local\_n, int partner)

{

int ai = my\_first + local\_n - 1;

int bi = my\_first + local\_n - 1;

int xi = (partner+1)\*local\_n - 1;

for (int i = 0; i < local\_n; ++i)

{

if (l\_a[ai] >= l\_a[xi]) { l\_b[bi--] = l\_a[ai--]; }

else { l\_b[bi--] = l\_a[xi--]; }

}

}

//Барьер

void barrier(void)

{

if(pthread\_mutex\_lock(&bar\_mutex) != 0)

{

fprintf(stderr, "MUTEX LOCK ERROR\n");

exit(-1);

}

++bar\_count;

if (bar\_count == thread\_count)

{

bar\_count = 0;

if(pthread\_cond\_broadcast(&bar\_cond) != 0)

{

fprintf(stderr, "COND BROADCAST ERROR\n");

exit(-1);

}

}

else { while (pthread\_cond\_wait(&bar\_cond, &bar\_mutex) != 0); }

if(pthread\_mutex\_unlock(&bar\_mutex) != 0)

{

fprintf(stderr, "MUTEX UNLOCK ERROR\n");

exit(-1);

}

}

//Функция сравнения

int compare(void const\* x\_p, void const\* y\_p)

{

int x = \*((int\*)x\_p);

int y = \*((int\*)y\_p);

if (x < y) { return -1; }

else if (x == y) { return 0; }

else { return 1; }

}

**bitonic\_sort.h:**

#ifndef \_\_BITONICSORT\_

#define \_\_BITONICSORT\_

void\* bitonic\_sort(void\*);

void bitonic\_sort\_incr(int,int,int,int,int);

void bitonic\_sort\_decr(int,int,int,int,int);

void merge\_split\_lo(int,int,int,int);

void merge\_split\_hi(int,int,int,int);

void barrier(void);

int compare(void const\*, void const\*);

#endif

**Пример работы**

//миллион чисел

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 1 1000000 g

Elapsed time = 292 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 2 1000000 g

Elapsed time = 153 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 4 1000000 g

Elapsed time = 89 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 8 1000000 g

Elapsed time = 68 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 16 1000000 g

Elapsed time = 79 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 32 1000000 g

Elapsed time = 74 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 64 1000000 g

Elapsed time = 101 milliseconds

//десять миллионов чисел

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 1 10000000 g

Elapsed time = 3275 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 2 10000000 g

Elapsed time = 1652 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 4 10000000 g

Elapsed time = 845 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 8 10000000 g

Elapsed time = 564 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 16 10000000 g

Elapsed time = 615 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 32 10000000 g

Elapsed time = 641 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 64 10000000 g

Elapsed time = 825 milliseconds

//сто миллионов чисел

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 1 100000000 g

Elapsed time = 27054 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 2 100000000 g

Elapsed time = 16445 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 4 100000000 g

Elapsed time = 8981 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 8 100000000 g

Elapsed time = 6003 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 16 100000000 g

Elapsed time = 6251 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 32 100000000 g

Elapsed time = 6643 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 64 100000000 g

Elapsed time = 7195 milliseconds

//десять чисел, показать что сортировка работает

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 1 10 o < test\_02.txt

Enter the list

The input list is:

6 2 7 1 10 3 5 8 4 9

The sorted list is:

1 2 3 4 5 6 7 8 9 10

Elapsed time = 0 millisecond

//миллион чисел из файла

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 1 1000000 < test\_01.txt

Enter the list

Elapsed time = 287 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 2 1000000 < test\_01.txt

Enter the list

Elapsed time = 150 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 4 1000000 < test\_01.txt

Enter the list

Elapsed time = 81 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 8 1000000 < test\_01.txt

Enter the list

Elapsed time = 55 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 16 1000000 < test\_01.txt

Enter the list

Elapsed time = 62 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 32 1000000 < test\_01.txt

Enter the list

Elapsed time = 65 milliseconds

pe4eniks@pe4eniks-HP-Laptop-14-dk0xxx:~$ ./a.out 64 1000000 < test\_01.txt

Enter the list

Elapsed time = 101 milliseconds

**Вывод**

На СИ можно писать программы использующие распараллеливание, ускорение и эффективность алгоритма зависит от количества данных и количества потоков, на маленьких данных лучше работают алгоритмы использующие небольшое количество потоков, чем больше данных тем лучше начинают работать алгоритмы с бОльшим числом потоков, однако улучшение не может наблюдаться бесконечно, поэтому в какой-то момент с увеличением числа потоков ускорение не увеличится, а эффективность будет падать.