Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №5

на тему

**УПРАВЛЕНИЕ ПОТОКАМИ, СРЕДСТВА СИНХРОНИЗАЦИИ**

|  |
| --- |
| Выполнил: студент гр. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Гриценко Н.Ю. |

Минск 2025

**СОДЕРЖАНИЕ**

[1 Постановка задачи и программа для её решения 3](#_Toc193113658)

[1.1 Постановка задачи 3](#_Toc193113659)

[1.2 Демонстрация работы программы 3](#_Toc193113660)

[Заключение 4](#_Toc193113661)

[Список использованных источников 5](#_Toc193113662)

[Приложение А (справочное) Исходный код 6](#_Toc193113663)

**1 ПОСТАНОВКА ЗАДАЧИ И ПРОГРАММА ДЛЯ ЕЁ РЕШЕНИЯ**

**1.1 Постановка задачи**

В рамках лабораторной работы по теме «Управление потоками и средства синхронизации» необходимо разработать многопоточную программу для сортировки массива данных. Программа должна эффективно обрабатывать достаточно большой массив, разбивая его на несколько частей, которые будут сортироваться параллельно отдельными потоками. После сортировки каждого фрагмента, результаты должны быть собраны в окончательный отсортированный массив. Пользователь должен иметь возможность задавать размер массива и количество потоков, при этом следует ограничить количество потоков, чтобы избежать перегрузки системы и обеспечить удобство отображения результатов. В процессе реализации программы требуется учитывать особенности работы с потоками, включая создание, завершение и синхронизацию потоков с помощью библиотек *pthread*. В качестве результата лабораторной работы необходимо представить сведения о времени выполнения программы для различных конфигураций, а также минимальный протокол выполнения, в котором будут отражены ключевые этапы работы программы и результаты тестирования.

**1.2 Демонстрация работы программы**

Программный код предоставлен в приложении А.

Для реализации поставленной задачи была создана программа, принимающая на вход количество элементов массива и количество потоков на которые нужно разбить выполнение, результатом выполнения является время, которое программа выполняла сортировку (рисунок 1.1).

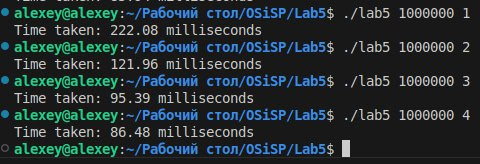


Рисунок 1.1 – Выполнение программы

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была разработана многопоточная программа для сортировки массива данных с использованием библиотеки *pthread*. Программа эффективно распределяет задачи сортировки между несколькими потоками, каждый из которых обрабатывает свою часть массива. Реализованный алгоритм включает разбиение данных на фрагменты, их параллельную сортировку с помощью функции *qsort* [2], а затем слияние отсортированных фрагментов в окончательный массив.

Результаты работы программы показали, что использование многопоточности позволяет значительно сократить время выполнения сортировки по сравнению с однопоточной реализацией, особенно при увеличении размера массива. Также в ходе тестирования были выявлены ключевые аспекты, касающиеся синхронизации потоков и управления ими, что подтвердило важность правильного использования механизмов синхронизации, таких как мьютексы и барьеры, для обеспечения корректности выполнения многопоточных программ.

Проведенные эксперименты продемонстрировали, что эффективность работы программы зависит от числа потоков и размера обрабатываемого массива. В дальнейшем можно рассмотреть оптимизацию алгоритма с учетом особенностей конкретных задач и архитектуры системы, что может привести к еще большему ускорению обработки данных. В целом, лабораторная работа позволила углубить понимание принципов работы с потоками и механизмами синхронизации в операционных системах, а также развить навыки проектирования и реализации многопоточных приложений.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Многопоточность [Электронный ресурс]. – Режим доступа: https://metanit.com/c/tutorial/11.1.php.

[2] qsort() Function in C [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/qsort-function-in-c/.

ПРИЛОЖЕНИЕ А  
(справочное)  
Исходный код

Листинг А.1 – Главная программа

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <time.h>

#include <limits.h>

#include <string.h>

typedef struct {

int\* data;

int start;

int end;

} ThreadArgs;

int compare\_int(const void\* a, const void\* b) {

return (\*(int\*)a - \*(int\*)b);

}

void\* thread\_sort(void\* arg) {

ThreadArgs\* args = (ThreadArgs\*)arg;

qsort(args->data + args->start, args->end - args->start, sizeof(int), compare\_int);

return NULL;

}

void merge(int\* data, int\* temp, int num\_threads, const int\* starts, const int\* ends, int size) {

int\* current\_pos = malloc(num\_threads \* sizeof(int));

for (int i = 0; i < num\_threads; i++) {

current\_pos[i] = starts[i];

}

int index = 0;

while (index < size) {

int min\_val = INT\_MAX;

int min\_thread = -1;

for (int i = 0; i < num\_threads; i++) {

if (current\_pos[i] < ends[i]) {

int val = data[current\_pos[i]];

if (val < min\_val) {

min\_val = val;

min\_thread = i;

}

}

}

if (min\_thread == -1) break;

temp[index++] = min\_val;

current\_pos[min\_thread]++;

}

memcpy(data, temp, size \* sizeof(int));

free(current\_pos);

}

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

if (argc != 3) {

fprintf(stderr, "Usage: %s <array\_size> <num\_threads>\n", argv[0]);

return 1;

}

int size = atoi(argv[1]);

int num\_threads = atoi(argv[2]);

if (num\_threads <= 0 || size <= 0) {

fprintf(stderr, "Invalid arguments\n");

return 1;

}

int\* data = malloc(size \* sizeof(int));

int\* temp = malloc(size \* sizeof(int));

if (!data || !temp) {

perror("malloc failed");

return 1;

}

srand(time(NULL));

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

data[i] = rand() % 1000;

}

ThreadArgs\* thread\_args = malloc(num\_threads \* sizeof(ThreadArgs));

int\* starts = malloc(num\_threads \* sizeof(int));

int\* ends = malloc(num\_threads \* sizeof(int));

int chunk\_size = size / num\_threads;

int remainder = size % num\_threads;

int current\_start = 0;

for (int i = 0; i < num\_threads; i++) {

int current\_end = current\_start + chunk\_size + (i < remainder ? 1 : 0);

starts[i] = current\_start;

ends[i] = current\_end;

thread\_args[i].data = data;

thread\_args[i].start = current\_start;

thread\_args[i].end = current\_end;

current\_start = current\_end;

}

pthread\_t\* threads = malloc(num\_threads \* sizeof(pthread\_t));

struct timespec start\_time, end\_time;

clock\_gettime(CLOCK\_MONOTONIC, &start\_time);

for (int i = 0; i < num\_threads; i++) {

if (pthread\_create(&threads[i], NULL, thread\_sort, &thread\_args[i]) != 0) {

perror("pthread\_create");

return 1;

}

}

for (int i = 0; i < num\_threads; i++) {

if (pthread\_join(threads[i], NULL) != 0) {

perror("pthread\_join");

return 1;

}

}

merge(data, temp, num\_threads, starts, ends, size);

clock\_gettime(CLOCK\_MONOTONIC, &end\_time);

for (int i = 0; i < size - 1; i++) {

if (data[i] > data[i + 1]) {

printf("Array is not sorted correctly at index %d\n", i);

break;

}

}

long long elapsed\_ns = (end\_time.tv\_sec - start\_time.tv\_sec) \* 1000000000LL + (end\_time.tv\_nsec - start\_time.tv\_nsec);

printf("Time taken: %.2f milliseconds\n", elapsed\_ns / 1000000.0);

free(data);

free(temp);

free(thread\_args);

free(starts);

free(ends);

free(threads);

return 0;

}

Листинг А.2 – Makefile

CC=gcc

CFLAGS=-Wall -Wextra -pthread

TARGET=lab5

all: $(TARGET)

$(TARGET): $(TARGET).c

$(CC) $(CFLAGS) -o $@ $^

clean:

rm -f $(TARGET)