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

ОТЧЕТ

к лабораторной работе №5

на тему

**УПРАВЛЕНИЕ ПОТОКАМИ, СРЕДСТВА СИНХРОНИЗАЦИИ**

Студент М. Ю. Пухов

Преподаватель С. И. Сиротко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Изучение подсистемы потоков (pthread), основных особенностей функционирования и управления, средств взаимодействия потоков.  
Практическое проектирование, реализация и отладка программ с параллельными взаимодействующими (конкурирующими) потоками.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Поток (thread) – это единица выполнения внутри одного процесса, которая использует общие ресурсы процесса, но имеет собственный стек и регистры.

Основным преимуществом многопоточного программирования является параллельное выполнение кода для лучшего использования ресурсов многоядерных процессоров.

Для организации потоков в Linux используется стандартная библиотека pthread. Основные функции для работы с потоками:

* pthread\_create() – создание нового потока
* pthread\_exit() – завершение потока
* pthread\_join() – ожидание завершения потока
* pthread\_detach() – отсоединение потока от управляющего потока

Для синхронизации потоков применяются мьютексы, семафоры, условные переменные. Они позволяют избежать конфликтов при работе с разделяемыми данными.

В задании реализована параллельная сортировка слиянием. Массив делится на фрагменты, каждый из которых сортируется отдельным потоком. Затем фрагменты сливаются в результирующий отсортированный массив.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

Программа была успешно выполнена с использованием следующих параметров: 10 2 1.

Время выполнения параллельной сортировки слиянием без выделения памяти: 1002341 наносекунда.

Общее время выполнения параллельной сортировки слиянием (включая выделение памяти): 1411282 наносекунды.

Время выполнения сортировки с использованием стандартной функции qsort: 8833 наносекунды.

Отсортированный массив, полученный параллельной сортировкой слиянием: 15 21 35 49 77 83 86 86 92 93

Отсортированный массив, полученный сортировкой с использованием qsort: 15 21 35 49 77 83 86 86 92 93

Программа автоматически проверила корректность сортировки, сравнивая каждый элемент в отсортированных массивах. Результат: отсутствие расхождений.

Таким образом, программа работает корректно, и отсортированные массивы совпадают, что подтверждает успешное выполнение сортировки слиянием в параллельном режиме. Результат выполнения ниже (рисунок 1).

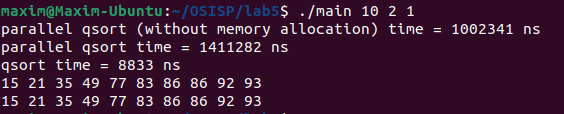


Рисунок 1 – Результат выполнения

# ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была реализована параллельная сортировка массива целых чисел с использованием технологии многопоточного программирования в Linux.

Были изучены базовые возможности библиотеки потоков pthread, включая создание и завершение потоков, ожидание завершения потока, передачу аргументов в поток.

Реализован алгоритм сортировки слиянием с распараллеливанием на этапах сортировки фрагментов и их последующего слияния. Количество потоков и размер массива задаются динамически.

Проведено тестирование для различных размеров входных данных и количества потоков. Полученная параллельная реализация демонстрирует сокращение времени сортировки по сравнению с последовательным алгоритмом.

Данная программа может служить основой для дальнейшего исследования эффективности распараллеливания алгоритмов сортировки. Перспективными направлениями развития являются оптимизация разбиения на фрагменты и автоматический выбор оптимального числа потоков.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

# A Practical Introduction to Parallel Computing [Электронный ресурс]. – Режим доступа: https://computing.llnl.gov/tutorials/parallel\_comp/

# OpenMP Documentation [Электронный ресурс]. – Режим доступа: https://www.openmp.org/specifications/

# Intel Threading Building Blocks [Электронный ресурс]. – Режим доступа: https://software.intel.com/content/www/us/en/develop/documentation/tbb-documentation.html

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *sort.c*

#include "sort.h"

#include <stdio.h>

#include <stdlib.h>

// Функция для слияния двух подмассивов

void merge(size\_t l, size\_t q, size\_t r, int\* const A, int\* L, int\* R) {

size\_t n1 = q - l + 1;

size\_t n2 = r - q;

// Копирование элементов во временные массивы L и R

for (size\_t i = 0; i < n1; i++) {

L[i] = A[l + i];

}

for (size\_t i = 0; i < n2; i++) {

R[i] = A[q + 1 + i];

}

// Добавление бесконечности в конец временных массивов

L[n1] = \_\_INT\_MAX\_\_;

R[n2] = \_\_INT\_MAX\_\_;

int i = 0;

int j = 0;

// Слияние отсортированных подмассивов

for (size\_t k = l; k <= r; k++) {

if (L[i] <= R[j]) {

A[k] = L[i++];

} else {

A[k] = R[j++];

}

}

}

// Функция для сравнения элементов при использовании qsort

int comp(const void\* i, const void\* j) {

return \*(int\*)i - \*(int\*)j;

}

// Обертка для qsort, вызываемая из потока

void\* qsort\_wrapper(void\* args) {

// Преобразование аргументов

struct SortArgs\* merge\_args = args;

int l = merge\_args->l;

int r = merge\_args->r;

int\* array = merge\_args->A;

// Вызов qsort для сортировки подмассива

if (l < r) {

qsort(array + l, r - l + 1, sizeof(int), comp);

}

return NULL;

}

Листинг 2 – Файл *sort.h*

*#ifndef SORT\_H*

*#define SORT\_H*

*#include <stddef.h>*

*// Макросы для определения максимума и минимума*

*#define MAX(a, b) ((a) > (b) ? (a) : (b))*

*#define MIN(a, b) ((a) < (b) ? (a) : (b))*

*// Прототип функции для слияния двух подмассивов*

*void merge(size\_t l, size\_t q, size\_t r, int\* const A, int\* L, int\* R);*

*// Прототип функции-обертки для qsort, вызываемой из потока*

*void\* qsort\_wrapper(void\* args);*

*// Прототип функции для сравнения элементов при использовании qsort*

*int comp(const void\* i, const void\* j);*

*// Структура для передачи аргументов в функцию-обертку*

*struct SortArgs {*

*int l;*

*int r;*

*int\* A;*

*};*

*#endif // SORT\_H*

Листинг 3 – Файл *main.c*

*#include <pthread.h>*

*#include <stdlib.h>*

*#include <stdio.h>*

*#include <unistd.h>*

*#include <ctype.h>*

*#include <string.h>*

*#include <math.h>*

*#include <time.h>*

*#include <stdint.h>*

*#include "sort.h"*

*// Функция для вывода массива*

*void print(const int\* const array, size\_t array\_size) {*

*for (size\_t i = 0; i < array\_size; i++) {*

*printf("%d ", array[i]);*

*}*

*printf("\n");*

*}*

*// Функция для параллельной сортировки слиянием*

*void parallel\_merge\_sort(int\* array, size\_t n, size\_t threads\_num) {*

*// Выделение памяти под массив потоков и структуры с аргументами*

*pthread\_t\* threads = (pthread\_t\*)malloc(threads\_num \* sizeof(pthread\_t));*

*struct SortArgs\* args =*

*(struct SortArgs\*)malloc(threads\_num \* sizeof(struct SortArgs));*

*int\* L = (int\*)malloc(n \* sizeof(int));*

*int\* R = (int\*)malloc(n \* sizeof(int));*

*// Измерение времени начала сортировки*

*uint64\_t diff;*

*struct timespec start, end;*

*clock\_gettime(CLOCK\_MONOTONIC\_RAW , &start);*

*// Распределение диапазонов по потокам для сортировки qsort*

*size\_t range\_size =*

*((size\_t)ceil((double)n / threads\_num) \* threads\_num) / threads\_num;*

*for (size\_t i = 0; i < threads\_num; i++) {*

*args[i].l = i \* range\_size;*

*args[i].r = MIN(n - 1, (i + 1) \* range\_size - 1);*

*args[i].A = array;*

*pthread\_create(&threads[i], NULL, qsort\_wrapper, (void\*)&args[i]);*

*}*

*// Ожидание завершения сортировки qsort в каждом потоке*

*for (size\_t i = 0; i < threads\_num; i++) {*

*pthread\_join(threads[i], NULL);*

*}*

*// Слияние подмассивов в один отсортированный массив*

*while (range\_size < n) {*

*for (size\_t i = 0; i < (size\_t)ceil((double)n / (range\_size << 1));*

*i++) {*

*size\_t l = i \* (range\_size << 1),*

*q = MIN(n - 1, i \* (range\_size << 1) + range\_size - 1),*

*r = MIN(n - 1, (i + 1) \* (range\_size << 1) - 1);*

*merge(l, q, r, array, L, R);*

*}*

*range\_size <<= 1;*

*}*

*// Измерение времени окончания сортировки*

*clock\_gettime(CLOCK\_MONOTONIC\_RAW , &end);*

*diff = BILLION \* (end.tv\_sec - start.tv\_sec) + end.tv\_nsec - start.tv\_nsec;*

*printf("parallel qsort (without memory allocation) time = %ld ns\n", diff);*

*// Освобождение выделенной памяти*

*free(threads);*

*free(args);*

*free(L);*

*free(R);*

*}*

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

*// Проверка наличия правильного числа аргументов командной строки*

*if (argc != 4) {*

*printf("Usage: ./program <array\_size> <num\_threads> <print? (0, 1)>\n");*

*return 1;*

*}*

*// Извлечение параметров из аргументов командной строки*

*size\_t size = atoi(argv[1]);*

*int threads\_num = atoi(argv[2]);*

*int need\_print = atoi(argv[3]);*

*// Проверка валидности введенных параметров*

*if (size <= 0 || threads\_num <= 0 || (need\_print != 0 && need\_print != 1)) {*

*printf(*

*"Invalid input. Array size and number of threads must be positive "*

*"integers and third parameter 0 or 1.\n");*

*return 1;*

*}*

*// Ограничение числа потоков максимальным значением*

*if (threads\_num > MAX\_THREADS) {*

*printf(*

*"Number of threads exceeds the maximum limit. Using maximum "*

*"threads: %d\n",*

*MAX\_THREADS);*

*threads\_num = MAX\_THREADS;*

*}*

*// Выделение памяти под массивы*

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

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

*// Заполнение массивов случайными значениями*

*for (size\_t i = 0; i < size; i++) {*

*A[i] = rand() % 100;*

*B[i] = A[i];*

*}*

*// Измерение времени начала сортировки и вызов функции параллельной сортировки*

*uint64\_t diff;*

*struct timespec start, end;*

*clock\_gettime(CLOCK\_MONOTONIC\_RAW , &start);*

*parallel\_merge\_sort(A, size, threads\_num);*

*clock\_gettime(CLOCK\_MONOTONIC\_RAW , &end);*

*diff = BILLION \* (end.tv\_sec - start.tv\_sec) + end.tv\_nsec - start.tv\_nsec;*

*printf("parallel qsort time = %ld ns\n", diff);*

*// Измерение времени сортировки с использованием qsort и вывод результатов*

*clock\_gettime(CLOCK\_MONOTONIC\_RAW , &start);*

*qsort(B, size, sizeof(int), comp);*

*clock\_gettime(CLOCK\_MONOTONIC\_RAW , &end);*

*diff = BILLION \* (end.tv\_sec - start.tv\_sec) + end.tv\_nsec - start.tv\_nsec;*

*printf("qsort time = %ld ns\n", diff);*

*// Вывод массивов, если необходимо*

*if (need\_print) {*

*print(A, size);*

*print(B, size);*

*}*

*// Проверка корректности сортировки*

*for (size\_t i = 0; i < size; i++) {*

*if (A[i] != B[i]) {*

*printf("WRONG...");*

*break;*

*}*

*}*

*// Освобождение выделенной памяти*

*free(A);*

*free(B);*

*return 0;*

*}*