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**

Выполнил:

студент гр. 153503

Кончик Д.С.

Проверил:

Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146836467)

[2 Теоретические сведения 4](#_Toc146836468)

[3 Полученные результаты 5](#_Toc146836469)

[Выводы 6](#_Toc146836470)

[Список использованных источников 7](#_Toc146836471)

[Приложение А (обязательное) листинг кода 8](#_Toc146836472)

# **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.

Написать программу, реализующую многопоточную обработку достаточно большого массива данных, например его сортировку. Количество потоков (в т.ч. единственный) и размер массива задаются пользователем. Спланировать и обеспечить тестирование (демонстрацию) выполнения.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ** 

Процесс – экземпляр программы во время выполнения, независимый объект, которому выделены системные ресурсы (например, процессорное время и память). Каждый процесс выполняется в отдельном адресном пространстве: один процесс не может получить доступ к переменным и структурам данных другого. Если процесс хочет получить доступ к чужим ресурсам, необходимо использовать межпроцессное взаимодействие. Это могут быть конвейеры, файлы, каналы связи между компьютерами и многое другое.

Поток использует то же самое пространства стека, что и процесс, а множество потоков совместно используют данные своих состояний. Как правило, каждый поток может работать (читать и писать) с одной и той же областью памяти, в отличие от процессов, которые не могут просто так получить доступ к памяти другого процесса. У каждого потока есть собственные регистры и собственный стек, но другие потоки могут их использовать.

Поток – определенный способ выполнения процесса. Когда один поток изменяет ресурс процесса, это изменение сразу же становится видно другим потокам этого процесса [1].

Семафор – это переменная особого типа, которая может изменяться с положительным или отрицательным приращением, но обращение к переменной в ответственный момент всегда атомарно даже в многопоточных программах. Это означает, что, если два или несколько потоков в программе пытаются изменить значение семафора, система гарантирует, что все операции будут на самом деле выполняться одна за другой.

Мьютекс (взаимоисключение, mutex) – примитив синхронизации, устанавливающийся в особое сигнальное состояние, когда не занят каким-либо потоком. Только один поток владеет этим объектом в любой момент времени, отсюда и название таких объектов – одновременный доступ к общему ресурсу исключается.

Спин-блокировки представляют собой чрезвычайно низкоуровневое средство синхронизации, предназначенное в первую очередь для применения в многопроцессорной конфигурации с разделяемой памятью. Они обычно реализуются как атомарно устанавливаемое булево значение. Аппаратура поддерживает подобные блокировки командами вида «проверить и установить» [2].

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В результате выполнения лабораторной работы был написана программа, реализующая многопоточную обработку достаточно большого массива данных, а именно его сортировку.

Программа при старте запрашивает количество потоков и размер массива. Далее рандомно заполняет массив, запускает таймер. После этого разбивает исходный массив на количество потоков и в каждом потоке сортирует соответствующую часть массива. Далее соединяет отсортированные части массива в один и останавливает таймер и выводит результат на экран(рисунок 1).

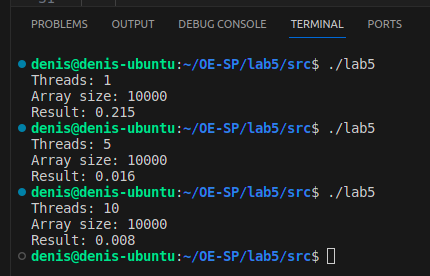


Рисунок 1 – Результат работы программы

# **ВЫВОДЫ**

В результате выполнения лабораторной работы были изучены подсистемыы потоков (pthread), основные особенности функционирования и управления, средств взаимодействия потоков.

Написана программа, реализующая многопоточную обработку достаточно большого массива данных, а именно его сортировку. Количество потоков (в т.ч. единственный) и размер массива задаются пользователем.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Что такое процесс [Электронный ресурс]. – Режим доступа: https://tproger.ru/problems/what-is-the-difference-between-threads-and-processes.
2. Методы синхронизации потоков [Электронный ресурс]. – Режим доступа: https://science-pedagogy.ru/ru/article/view?id=1962.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг 1 **–** Файл *main.c*:

*#include* <stdio.h>  
*#include* <stdlib.h>  
*#include* <pthread.h>  
*#include* <sys/time.h>  
*#include* <limits.h>  
  
*typedef struct* {  
 *int*\* array;  
 *int* start;  
 *int* end;  
} arr\_part;  
  
*void* bubble\_sort(arr\_part\* arr\_part);  
*void* merge\_arrays(*int*\* array, *int* array\_size, *int* num\_threads, *int* part\_size);  
  
*int* main() {  
 *int* num\_threads;  
 printf("Threads: ");  
 scanf("%d", &num\_threads);  
 *if* (num\_threads < 1) {  
 *return* 0;  
 }  
 *int* array\_size;  
 printf("Array size: ");  
 scanf("%d", &array\_size);  
 *int* \*array = malloc(array\_size \* *sizeof*(*int*));  
  
 srand(time(NULL));  
 *for* (*int* i = 0; i < array\_size; i++)  
 array[i] = rand() % 100000;  
  
 *int* part\_size = array\_size / num\_threads;  
 pthread\_t threads[num\_threads];  
 arr\_part parts[num\_threads];  
  
 *struct* timeval start\_time, end\_time;  
 gettimeofday(&start\_time, NULL);  
  
 *for* (*int* i = 0; i < num\_threads; i++) {  
 parts[i].array = array;  
 parts[i].start = i \* part\_size;  
 parts[i].end = (i == num\_threads - 1) ? array\_size - 1 : (i + 1) \* part\_size - 1;  
 pthread\_create(&threads[i], NULL, (*void* \*(\*)(*void* \*)) bubble\_sort, &parts[i]);  
 }  
  
 *for* (*int* i = 0; i < num\_threads; i++) {  
 pthread\_join(threads[i], NULL);  
 }  
  
 merge\_arrays(array, array\_size, num\_threads, part\_size);  
  
 gettimeofday(&end\_time, NULL);  
 *double* execution\_time = (end\_time.tv\_sec - start\_time.tv\_sec) + (end\_time.tv\_usec - start\_time.tv\_usec) / 1e6;  
  
 printf("Result: %.3f\n", execution\_time);  
 free(array);  
 *return* 0;  
}  
  
  
*void* bubble\_sort(arr\_part\* arr\_part) {  
 *for* (*int* i = arr\_part->start; i <= arr\_part->end; i++) {  
 *for* (*int* j = i + 1; j <= arr\_part->end; j++) {  
 *if* (arr\_part->array[i] > arr\_part->array[j]) {  
 *int* temp = arr\_part->array[i];  
 arr\_part->array[i] = arr\_part->array[j];  
 arr\_part->array[j] = temp;  
 }  
 }  
 }  
  
 pthread\_exit(NULL);  
}  
  
*void* merge\_arrays(*int*\* array, *int* array\_size, *int* num\_threads, *int* part\_size) {  
 *int* temp[array\_size];  
 *int* index[num\_threads];  
  
 *for* (*int* i = 0; i < num\_threads; ++i) {  
 index[i] = i \* part\_size;  
 }  
  
 *for* (*int* i = 0; i < array\_size; ++i) {  
 *int* min\_val = INT\_MAX;  
 *int* min\_thread = -1;  
  
 *for* (*int* j = 0; j < num\_threads; ++j) {  
 *if* (index[j] < (j + 1) \* part\_size && array[index[j]] < min\_val) {  
 min\_val = array[index[j]];  
 min\_thread = j;  
 }  
 }  
  
 temp[i] = min\_val;  
 ++index[min\_thread];  
 }  
  
 *for* (*int* i = 0; i < array\_size; ++i) {  
 array[i] = temp[i];  
 }  
}