

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
2 Теоретические сведения .....	4
3 Полученные результаты .....	5
Выводы .....	6
Список использованных источников .....	7
Приложение А (обязательное) листинг кода.....	8

## **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.

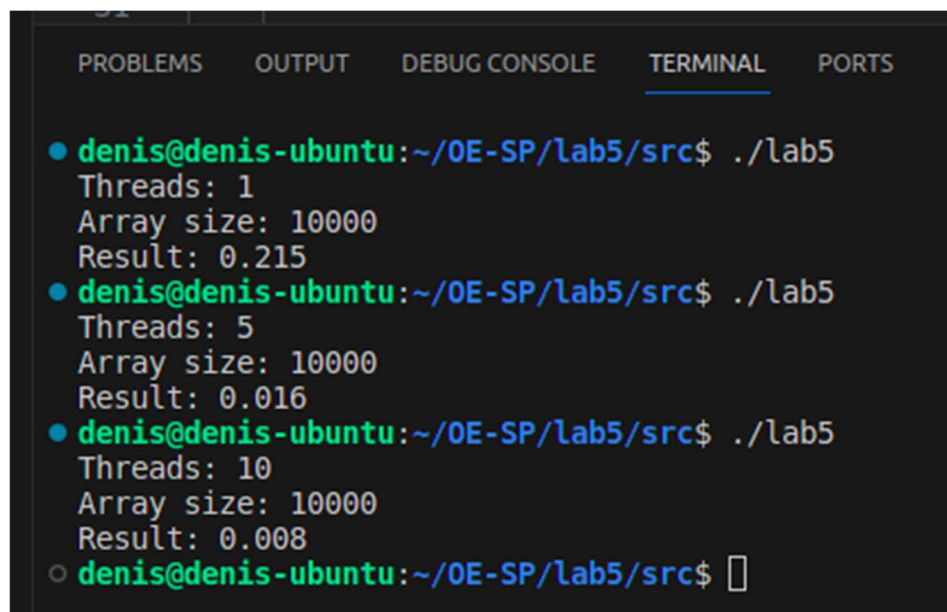
Мьютекс (взаимоисключение, mutex) – примитив синхронизации, устанавливающийся в особое сигнальное состояние, когда не занят каким-либо потоком. Только один поток владеет этим объектом в любой момент времени, отсюда и название таких объектов – одновременный доступ к общему ресурсу исключается.

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).



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● denis@denis-ubuntu:~/OE-SP/lab5/src$ ./lab5
  Threads: 1
  Array size: 10000
  Result: 0.215
● denis@denis-ubuntu:~/OE-SP/lab5/src$ ./lab5
  Threads: 5
  Array size: 10000
  Result: 0.016
● denis@denis-ubuntu:~/OE-SP/lab5/src$ ./lab5
  Threads: 10
  Array size: 10000
  Result: 0.008
○ denis@denis-ubuntu:~/OE-SP/lab5/src$ █
```

Рисунок 1 – Результат работы программы

## **ВЫВОДЫ**

В результате выполнения лабораторной работы были изучены подсистемы 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];
    }
}

```