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             А. Д. Филипеня

Проверил                          Н. Ю. Гриценко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc157722973)

[2 Краткие теоретические сведения 4](#_Toc157722974)

[3 Результаты выполнения лабораторной работы 5](#_Toc157722975)

[Выводы 6](#_Toc157722976)

[Список использованных источников 7](#_Toc157722977)

[Приложение А (обязательное) Листинг исходного кода 8](#_Toc157722978)

# **1 ПОСТАНОВКА ЗАДАЧИ**

Целью выполнения данной лабораторной работы является изучение подсистемы потоков, основных особенностей функционирования   
и управления, средств взаимодействия потоков. Также необходимо реализовать программу на языке программирования С, которая будет реализовывать многопоточную программу сортировки массива данных.

# **2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Поток выполнения – наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы   
не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса и его значения переменных, которые они имеют в любой момент времени.[1]

Каждый поток имеет свой уникальный целочисленный идентификатор, который называется TID. TID используется для идентификации конкретного потока в системе. Для главного потока процесса значения PID и TID совпадают, то есть фактически в качестве PID процесса выступает TID главного потока. Для всех остальных потоков процесса PID такое же,   
как и для главного потока, а значение TID уже индивидуально.[2]

Для головного «потока процесса» (группы) значения PID и TID совпадают. Точнее, в качестве PID процесса (группы) выступает TID головного потока. Для всех остальных потоков процесса (группы) значение PID такое же, как и для головного, а значения TID индивидуальные для каждого из потоков.

Продолжает действовать системный вызов getpid(), но добавляется также вызов gettid().

В многопоточных приложениях доступны традиционные средства синхронизации и обмена данными, такие как каналы, семафоры, мьютексы, разделяемая память.

Также одним из средств синхронизации являются барьеры. Особенность барьеров – счетчик синхронизируемых потоков: барьер «открывается» при достижении заданного числа потоков, ожидающих его открытия.

Для выполнения данной лабораторной работы были использованы сведения и концепции по управлению потоками. При помощи функции pthread\_create создавались новые потоки. При помощи функции pthread\_join блокировалось выполнение вызывающего эту функцию потока до тех пор, пока не завершатся все созданные раннее потоки. При помощи функции pthread\_exit завершается выполнение потоков и освобождаются ресурсы, связанные с ними.

# **3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ             РАБОТЫ**

В ходе лабораторной работы была создана программа, в которой реализуется многопоточная сортировка массива данных. Сортировка массива состоит из следующих стадий: разбиение массива на несколько фрагментов, сортировка каждого такого фрагмента в отдельном потоке и окончательная сборка. Количество потоков и размер массива задаются пользователем. Также отображаются сведения о времени выполнения сортировки массива. Для управления обработкой проекта используется makefile. Результат работы программы представлен на рисунке 3.1.

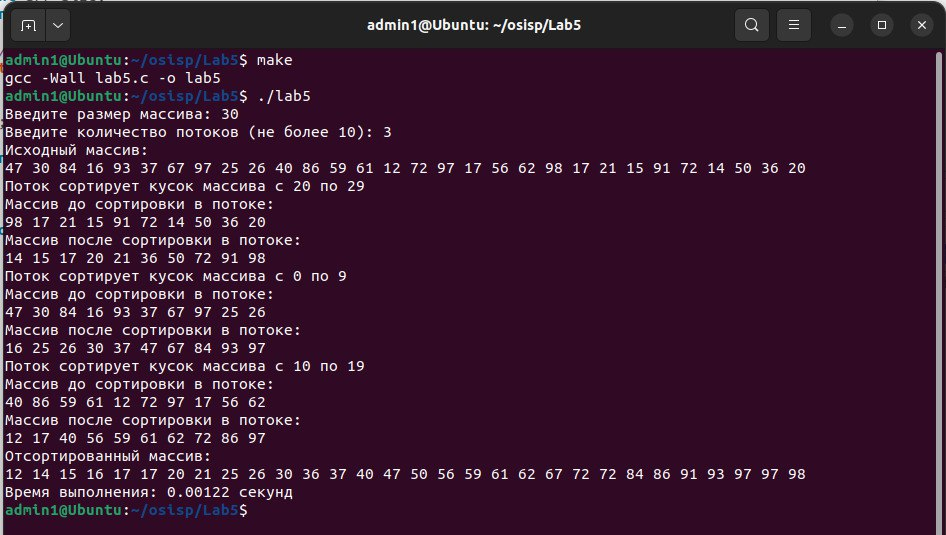


Рисунок 3.1 – Результат работы программы

Таким образом, в ходе лабораторной работы была реализована многопоточная программа сортировки массива данных.

# **ВЫВОДЫ**

В ходе лабораторной работы были изучены подсистема потоков, основные особенности функционирования и управления, средства взаимодействия потоков. Также была создана программа на языке программирования С, которая реализует многопоточную сортировку массива данных.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Процессы и потоки [Электронный ресурс]. – Режим доступа: <https://acm.bsu.by/wiki/Unix2019b/>. – Дата доступа: 18.02.2024.

[2] Архитектура Unix. Процессы и потоки [Электронный ресурс]. – Режим доступа: https://acm.bsu.by/wiki/Unix2018/. – Дата доступа: 19.02.2024.

## **ПРИЛОЖЕНИЕ А**

## **(обязательное)**

## **Листинг исходного кода**

Листинг 1 – Программный код lab5.c

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <unistd.h>

#include <sys/time.h>

#define MAX\_THREADS 10

int \*arr;

int arr\_size;

int num\_threads;

struct ThreadArgs {

int start;

int end;

};

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

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

}

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

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

printf("Поток сортирует кусок массива с %d по %d\n", args->start, args->end);

printf("Массив до сортировки в потоке:\n");

for (int i = args->start; i <= args->end; i++) {

printf("%d ", arr[i]);

}

printf("\n");

qsort(&arr[args->start], args->end - args->start + 1, sizeof(int), compare);

printf("Массив после сортировки в потоке:\n");

for (int i = args->start; i <= args->end; i++) {

printf("%d ", arr[i]);

}

printf("\n");

pthread\_exit(NULL);

}

void merge(int left, int mid, int right) {

int temp[right - left + 1];

int i = left, j = mid + 1, k = 0;

while (i <= mid && j <= right) {

if (arr[i] <= arr[j]) {

temp[k++] = arr[i++];

} else {

temp[k++] = arr[j++];

}

}

while (i <= mid) {

temp[k++] = arr[i++];

}

while (j <= right) {

temp[k++] = arr[j++];

}

for (i = left; i <= right; i++) {

arr[i] = temp[i - left];

}

}

void merge\_sort(int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

merge\_sort(left, mid);

merge\_sort(mid + 1, right);

merge(left, mid, right);

}

}

int main() {

struct timeval start\_time, end\_time;

pthread\_t threads[MAX\_THREADS];

struct ThreadArgs thread\_args[MAX\_THREADS];

int i;

printf("Введите размер массива: ");

scanf("%d", &arr\_size);

arr = (int\*)malloc(arr\_size \* sizeof(int));

printf("Введите количество потоков (не более %d): ", MAX\_THREADS);

scanf("%d", &num\_threads);

if (num\_threads > MAX\_THREADS || num\_threads <= 0) {

printf("Неверное количество потоков\n");

return 1;

}

printf("Исходный массив:\n");

srand(time(NULL));

for (i = 0; i < arr\_size; i++) {

arr[i] = rand() % 100;

printf("%d ", arr[i]);

}

printf("\n");

gettimeofday(&start\_time, NULL); // начало отсчета времени

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

thread\_args[i].start = i \* (arr\_size / num\_threads);

thread\_args[i].end = (i == num\_threads - 1) ? arr\_size - 1 : (i + 1) \* (arr\_size / num\_threads) - 1;

pthread\_create(&threads[i], NULL, thread\_sort, (void\*)&thread\_args[i]);

}

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

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

}

merge\_sort(0, arr\_size - 1);

gettimeofday(&end\_time, NULL); // конец отсчета времени

double execution\_time = (end\_time.tv\_sec - start\_time.tv\_sec) + (end\_time.tv\_usec - start\_time.tv\_usec) / 1000000.0;

printf("Отсортированный массив:\n");

for (i = 0; i < arr\_size; i++) {

printf("%d ", arr[i]);

}

printf("Время выполнения: %.5f секунд\n", execution\_time);

free(arr);

return 0;

}

Листинг 2 – Makefile

CC = gcc

CFLAGS = -Wall -pthread

all: lab5

lab5: lab5.c

$(CC) $(CFLAGS) lab5.c -o lab5

clean:

rm -f lab5