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 И СРЕДСТВА СИНХРОНИЗАЦИИ**

Студент В. В. Сапроненко

Преподаватель Н. Ю. Гриценко

Минск 2025

СОДЕРЖАНИЕ

[1 Постановка задачи](../../../../E:/Downloads/Telegram%20Desktop/%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%205%20(2).docx" \l "_Toc146631498) 3

[2 Краткие теоретические сведения](../../../../E:/Downloads/Telegram%20Desktop/%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%205%20(2).docx" \l "_Toc146631499) 4

[3 Результаты выполнения лабораторной работы](../../../../E:/Downloads/Telegram%20Desktop/%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%205%20(2).docx" \l "_Toc146631500) 5

[Выводы](../../../../E:/Downloads/Telegram%20Desktop/%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%205%20(2).docx" \l "_Toc146631501) 7

[Список использованных источников](../../../../E:/Downloads/Telegram%20Desktop/%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%205%20(2).docx" \l "_Toc146631502) 8

[Приложение А (обязательное) Листинг кода](../../../../E:/Downloads/Telegram%20Desktop/%D0%BB%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%205%20(2).docx" \l "_Toc146631503) 9

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является создание программы, реализующая обработку массива данных, например, его сортировку (алгоритм обработки должен допускать эффективное распараллеливание), с следующими основными этапами обработки:

– разбиение массива на несколько частей (фрагментов)

– сортировка каждого фрагмента отдельным потоком

– окончательная “сборка”

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Вычислительный поток (Thread) – ветвь алгоритма, последовательность (“поток”) машинных команд, выполняемая параллельно и независимо относительно других таких же последовательностей. Два основных подхода:

– Поток и процесс – две самостоятельные сущности: процесс – владелец ресурсов и контейнер потоков, поток – получатель времени процессора, участник планирования времени выполнения. Потоки одного процесса естественным образом разделяют и совместно используют его ресурсы, включая адресное пространство.

– Поток – “облегченный” процесс (большинство Unix-систем). Группа облегченных процессов совместно использует общий ресурсов, включая совместный доступ к адресному пространству.

Тип потока – joinable или detached – управляется атрибутом (при создании) или вызовом pthread\_detach(). Joinable – после завершения остается в состоянии “зомби” пока не будет запроса синхронизации с этим событием от другого потока. Этому потоку станут доступны код результата и другая информация о завершившемся. Detached (“отсоединенный”) – выполняется полностью самостоятельно (хотя и в рамках группы), после завершения удаляется из системы и не может участвовать в синхронизации.

Существует множество способов классификации различных форм IPC. Первое и наиболее распространенное разделение заключается в том, чтобы отделить методы, которые соответствуют модели передачи сообщений, от модели с общей памятью. При передаче сообщений всякий раз, когда требуется обмен данными, процесс вызывает ядро с запросом на отправку информации целевому процессу. То есть процесс пользовательского режима скопирует данные в буфер или другую структуру данных, а затем выполнит системный вызов с запросом на передачу данных. Как только ядро будет запущено, оно сначала скопирует переданные данные в свою собственную память. В какой-то момент целевой процесс также выполнит системный вызов для извлечения данных. Короче говоря, методы передачи сообщений требуют, чтобы ядро участвовало в каждом отдельном обмене данными. [1]

Методы работы с общей памятью принципиально отличаются от передачи сообщений. В общей памяти процессы изначально настраивают область своей виртуальной памяти для использования в IPC. Как только область установлена в рамках процесса, процесс выполняет системный вызов, чтобы сделать эту область общей. Другие процессы делают то же самое. После первоначального системного вызова для настройки общей памяти процессы могут выполнять чтение из области и запись в нее точно так же, как если бы они обращались к данным, не находящимся в общей памяти, а в своей собственной куче. То есть процесс мог бы выполнить запись в область путем разыменования указателя на нее. Затем эти данные автоматически отображаются в контексте другого процесса. Для чтения новых данных не требуется явного системного вызова. [1]

POSIX (Portable Operating System Interface) – Кроссплатформенная спецификация, поддерживаемая операционными системами UNIX и UNIX-подобными, например Linux. Буква X в названии изначально означала, что интерфейс “основан на UNIX”. [2]

System V – Спецификация, определяющая требования к операционной системе, чтобы она могла считаться UNIX.

Pthreads – аббревиатура от POSIX thread library. Библиотека языка программирования C, обеспечивающая кроссплатформенную поддержку многопоточности.

Семафор – это объект синхронизации, который используется для управления доступом к общим ресурсам. Представляет собой целое число с атомарными операциями для увеличения и уменьшения значения; если результат уменьшения значения отрицательный, текущий процесс блокируется до тех пор, пока другой процесс не увеличит значение. [2]

Мьютекс – Синоним lock; примитив синхронизации потоков POSIX, который можно использовать для обеспечения взаимоисключающего доступа к критической секции. От бинарного семафора его отличает следующее: предполагается, что бинарный семафор инкрементируют и декрементируют разные потоки, мьютекс же занимает и освобождает один и тот же поток. [2]

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы была создана программа, реализующая обработку массива данных. Результат работы этой программы представлен на рисунке ниже.

C:\Users\User\Desktop\image.png

Рисунок 3.1 – Результат работы программы

## ВЫВОДЫ

В ходе выполнения лабораторной работы была создана программа, реализующая обработку массива данных, (его сортировку. Был выбран алгоритм обработки допускающий эффективное распараллеливание), с следующими основными этапами обработки:

– разбиение массива на несколько частей (фрагментов)

– сортировка каждого фрагмента отдельным потоком

– окончательная “сборка”.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Glossary [Электронный ресурс]. – Режим доступа: https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/IPCModels.html. – Дата доступа: 02.04.2024.

[2] Glossary [Электронный ресурс]. – Режим доступа: https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/Glossary.html#term-portable-operating-system-interface. – Дата доступа: 02.04.2024.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**sort.c**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#define MAX\_THREAD 4

struct thread\_data {

int l;

int r;

int \*arr;

};

void merge(int arr[], int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

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

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

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

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void \*merge\_sort(void \*threadarg) {

struct thread\_data \*data = (struct thread\_data \*)threadarg;

int l = data->l;

int r = data->r;

int \*arr = data->arr;

if (l < r) {

int m = l + (r - l) / 2;

struct thread\_data data1, data2;

data1.l = l;

data1.r = m;

data1.arr = arr;

data2.l = m + 1;

data2.r = r;

data2.arr = arr;

pthread\_t threads[2];

pthread\_create(&threads[0], NULL, merge\_sort, (void \*)&data1);

pthread\_create(&threads[2], NULL, merge\_sort, (void \*)&data2);

pthread\_join(threads[0], NULL);

pthread\_join(threads[2], NULL);

merge(arr, l, m, r);

}

pthread\_exit(NULL);

}

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

int arr[] = {12, 11, 13, 5, 6, 7};

int arr\_size = sizeof(arr) / sizeof(arr[0]);

struct thread\_data data;

data.l = 0;

data.r = arr\_size - 1;

data.arr = arr;

pthread\_t sort\_thread;

pthread\_create(&sort\_thread, NULL, merge\_sort, (void \*)&data);

pthread\_join(sort\_thread, NULL);

printf("Sorted array: ");

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

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

printf("\n");

return 0;

}