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

ОТЧЁТ

к лабораторной работе №5

на тему

**УПРАВЛЕНИЕ ПОТОКАМИ, СРЕДСТВА СИНХРОНИЗАЦИИ**

Выполнил студент гр.153502 Толстой Д. В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc158209942)

[2 Описание функций программы 4](#_Toc158209943)

[Список использованных источников 5](#_Toc158209944)

[Приложение А (обязательное) Листинг кода 6](#_Toc158209945)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Многопоточная программа, реализующая обработку достаточно большого массива данных, например его сортировку (алгоритм обработки должен допускать эффективное распараллеливание).

Типовые стадии обработки (на примере сортировки):

1 разбиение массива на несколько частей (фрагментов).

2 сортировка каждого фрагмента отдельным потоком.

3 окончательная «сборка».

Количество потоков (в т.ч. единственный) и размер массива задаются пользователем. Количество потоков выбирается не слишком большое, чтобы оставалось удобным для отображения и не провоцировало перегрузку системы.

Результат – сведения о времени выполнения для конкретной конфигурации, минимальный протокол выполнения.

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Программа выполняет многопоточную сортировку массива целых чисел. Инициализирует переменные *n* и *num*\_*threads* для указания размера массива и количества потоков соответственно. Создает динамический массив *arr* размером *n*, который будет содержать случайно сгенерированные числа. Использует функцию *srand*(*time*(*NULL*)), чтобы проинициализировать генератор случайных чисел, а затем заполняет массив *arr* случайными числами от 0 до 99. Выводит на экран размер массива и количество потоков. Начинает отсчитывать время выполнения. Разбивает массив *arr* на равные части и запускает *num*\_*threads* потоков, каждый из которых сортирует свою часть массива, используя функцию *sort*. Ожидает завершения всех потоков с помощью *pthread*\_*join*. После завершения сортировки всех частей массива сливает отсортированные части с помощью функции *merge*. Заканчивает отсчет времени выполнения и выводит его на экран. Освобождает динамически выделенную память для массивов *arr*, *threads* и *args*.

Пример работы программы представлен на рисунке 1.

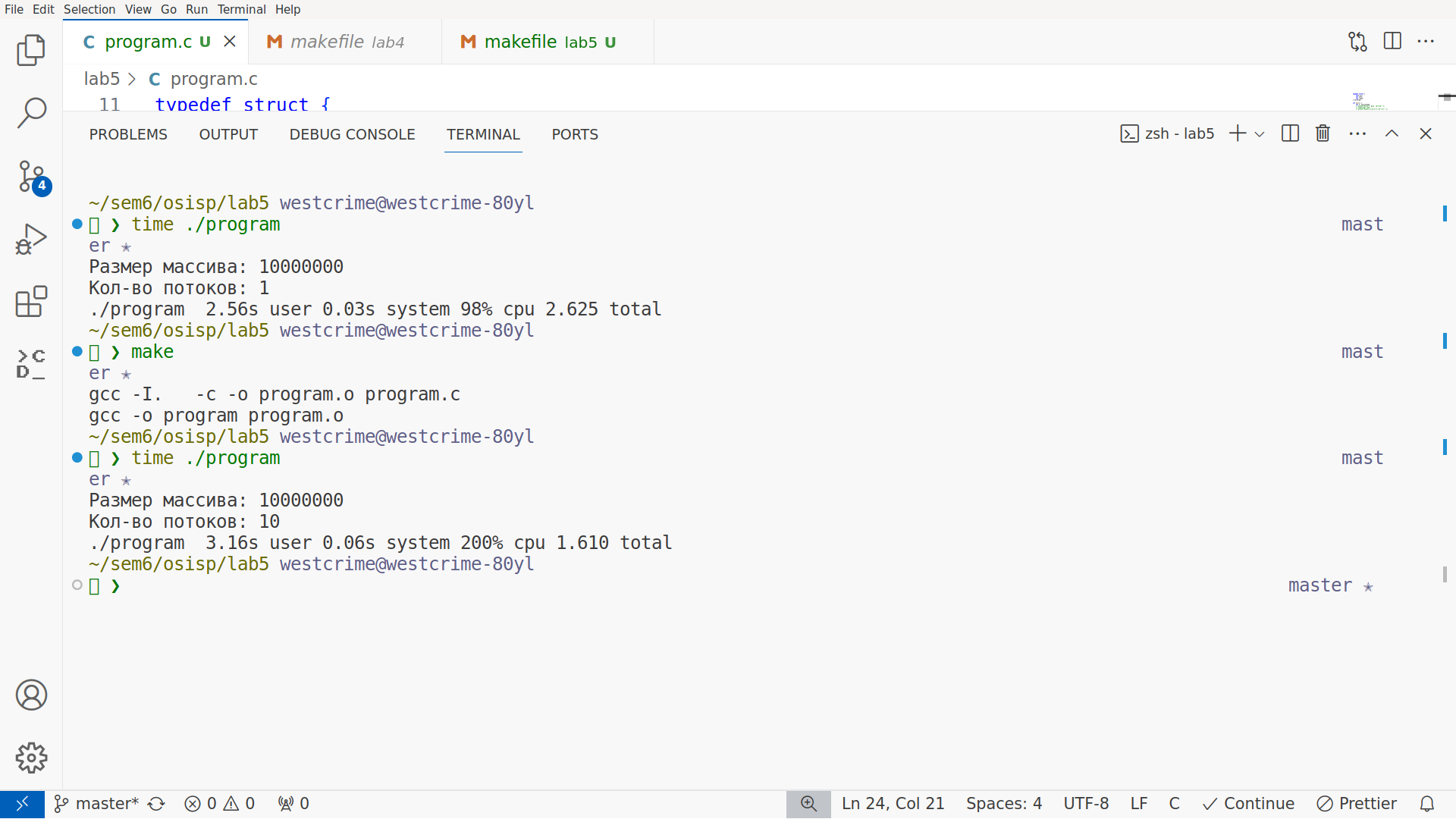


Рисунок 1 – Пример работы программы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Процессы UNIX [Электронный ресурс]. – Электронные данные. – Режим доступа: http://kharchuk.ru/home/15-unix-foundations/80-unix-processes

[2] pthreads() in C [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.geeksforgeeks.org/pthreads/

ПРИЛОЖЕНИЕ А

**(обязательное)   
Листинг кода**

program.c

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <time.h>

void\* sort(void\* arg);

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

int cmpfunc(const void\* a, const void\* b);

void print\_array(int\* arr, int size);

typedef struct {

    int\* arr;

    int left;

    int right;

} SortArgs;

int main() {

    int n, num\_threads;

    // printf("Введите размер массива: ");

    // scanf("%d", &n);

    // printf("Введите количество потоков: ");

    // scanf("%d", &num\_threads);

    n = 10000000;

    num\_threads = 10;

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

    pthread\_t\* threads = (pthread\_t\*)malloc(num\_threads \* sizeof(pthread\_t));

    SortArgs\* args = (SortArgs\*)malloc(num\_threads \* sizeof(SortArgs));

    // Инициализация массива случайными числами

    srand(time(NULL));

    for (int i = 0; i < n; i++) {

        arr[i] = rand() % 100;

    }

    printf("Размер массива: %d\n", n);

    printf("Кол-во потоков: %d\n", num\_threads);

    // Засекаем время начала

    clock\_t start\_time = clock();

    // Разбиение массива и запуск потоков для сортировки

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

        args[i].arr = arr;

        args[i].left = i \* (n / num\_threads);

        args[i].right = (i + 1) \* (n / num\_threads) - 1;

        if (i == num\_threads - 1) args[i].right = n - 1; // Для последнего потока, если n не делится на num\_threads

        pthread\_create(&threads[i], NULL, sort, &args[i]);

    }

    // Ожидание завершения потоков

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

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

    }

    // Слияние отсортированных частей

    for (int i = 1; i < num\_threads; i++) {

        merge(arr, 0, (i \* (n / num\_threads)) - 1, (i + 1) \* (n / num\_threads) - 1);

    }

    // Засекаем время окончания

    clock\_t end\_time = clock();

    // Выводим затраченное время

    // double time\_spent = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;

    // printf("Время выполнения: %f секунд\n", time\_spent);

    // print\_array(arr, n);

    // Освобождаем ресурсы

    free(arr);

    free(threads);

    free(args);

    return 0;

}

// Функция для сортировки части массива

void\* sort(void\* arg) {

    SortArgs\* args = (SortArgs\*)arg;

    qsort(args->arr + args->left, args->right - args->left + 1, sizeof(int), cmpfunc);

    return NULL;

}

// Функция сравнения для qsort

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

    return (\*(int\*)a - \*(int\*)b);

}

// Функция для слияния двух частей массива

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

    int i, j, k;

    int n1 = m - l + 1;

    int n2 = r - m;

    // Создаем временные массивы

    int\* L = (int\*)malloc(n1 \* sizeof(int));

    int\* R = (int\*)malloc(n2 \* sizeof(int));

    // Копируем данные во временные массивы L[] и R[]

    for (i = 0; i < n1; i++)

        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)

        R[j] = arr[m + 1 + j];

    // Сливаем временные массивы обратно в arr[l..r]

    i = 0; // Индекс первого подмассива

    j = 0; // Индекс второго подмассива

    k = l; // Индекс слияния подмассивов

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        } else {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

    // Копируем оставшиеся элементы L[], если они есть

    while (i < n1) {

        arr[k] = L[i];

        i++;

        k++;

    }

    // Копируем оставшиеся элементы R[], если они есть

    while (j < n2) {

        arr[k] = R[j];

        j++;

        k++;

    }

    free(L);

    free(R);

}

void print\_array(int\* arr, int size) {

    for (int i = 0; i < size; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

}