Лабораторная работа №2

по курсу "Операционные системы"

Выполнил: Юнусов Рустам М8О-310Б-22. 5 Вариант

Преподаватель: Миронов Евгений Сергеевич

Задание:

Отсортировать массив целых чисел при помощи четно-нечетной сортировки Бетчера

Листинг программы

```
#include <stdbool.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

struct ThreadArgs {
  int *array;
  int start;
  int end;
};

bool bool_sort_array(int array[], int array_size) {
  for (int index_in_arr = 0; index_in_arr < array_size - 1;
index_in_arr++) {
    if (array[index_in_arr] > array[index_in_arr + 1]) {
      return false;
    }
  }
  return true;
}

void sort_even_odd(int array[], int array_size) {
  int even[array_size / 2];
  int odd[array_size / 2];
  int indexEven = 0;
  int indexOdd = 0;
  for (int index_in_arr = 0; index_in_arr < array_size; index_in_arr++)
{
    if (index_in_arr % 2 == 0) {
      even[indexEven++] = array[index_in_arr];
    } else {
      odd[indexOdd++] = array[index_in_arr];
```

```
    }
  }
  indexOdd = 0;
  for (int index_in_arr = 0; index_in_arr < array_size; index_in_arr++)
{
    if (index_in_arr < array_size / 2) {
      array[index_in_arr] = even[index_in_arr];
    } else {
      array[index_in_arr] = odd[indexOdd++];
    }
  }
}

void sort_array(int array[], int array_size) {
  while (!bool_sort_array(array, array_size)) {
    for (int index_in_arr = 0; index_in_arr < array_size - 1;
index_in_arr++) {
      if (array[index_in_arr] > array[index_in_arr + 1]) {
        int temp = array[index_in_arr + 1];
        array[index_in_arr + 1] = array[index_in_arr];
        array[index_in_arr] = temp;
      }
    }
  }
}

void array_division(int array[], int array_size) {
  bool bool_extra_number = false;
  if (array_size % 2 != 0) {
    array_size--;
    bool_extra_number = true;
  }

  if (array_size != 2) {

    sort_even_odd(array, array_size);
    int first_half[array_size / 2];
    int secondHalf[array_size / 2];
    int second_half = 0;
    int indexfirst_half = 0;

    for (int index_in_arr = 0; index_in_arr < array_size;
index_in_arr++) {
      if (index_in_arr < array_size / 2) {
        first_half[index_in_arr] = array[index_in_arr];
      } else {
        secondHalf[second_half++] = array[index_in_arr];
      }
    }

    array_division(first_half, array_size / 2);
    array_division(secondHalf, array_size / 2);
```

```
    second_half = 0;
    for (int index_in_arr = 0; index_in_arr < array_size;
index_in_arr++) {
      if (index_in_arr % 2 == 0) {
        array[index_in_arr] = first_half[indexfirst_half++];
      } else {
        array[index_in_arr] = secondHalf[second_half++];
      }
    }
    if (bool_extra_number) {
      array_size++;
      sort_array(array, array_size);
    } else {
      sort_array(array, array_size);
    }
  } else {
    if (array[0] > array[1]) {
      int temp = array[1];
      array[1] = array[0];
      array[0] = temp;
    }
  }
}

void *calculations(void *arg) {
  struct ThreadArgs *args = (struct ThreadArgs*)arg;
  int array_size = args->end - args->start + 1;
  array_division(args->array + args->start, array_size);
  sort_array(args->array + args->start, array_size);
  return NULL;
}

int main(int argc, char *argv[]) {
  int number_of_threads = 0;

  if (argc < 2) {
    printf("Enter number of threads\n");
    return 1;
  }

  number_of_threads = atoi(argv[1]);

  printf("Enter array length: ");
  int array_size;
  scanf("%d", &array_size);
  int *array = (int*)malloc(sizeof(int) * array_size);
  printf("Enter array elements: ");
  for (int i = 0; i < array_size; ++i) {
    scanf("%d", &array[i]);
  }

  printf("Number of threads: %d\n", number_of_threads);
```

```
    struct timeval start, end;
    gettimeofday(&start, NULL);

    pthread_t threads[number_of_threads];
    struct ThreadArgs args[number_of_threads];

    for (int i = 0; i < number_of_threads; i++)
    {
      args[i].array = array;
      args[i].start = i * (array_size / number_of_threads);
      args[i].end = (i + 1) * (array_size / number_of_threads) - 1;
      pthread_create(&threads[i], NULL, calculations, (void *)&args[i]);
    }

    for (int i = 0; i < number_of_threads; i++)
    {
      pthread_join(threads[i], NULL);
    }

    sort_array(array, array_size);

    gettimeofday(&end, NULL);

    double execution_time = (end.tv_sec - start.tv_sec) * 1000.0;
    execution_time += (end.tv_usec - start.tv_usec) / 1000.0;

    printf("Execution time: %lf\n", execution_time);

    printf("Sorted array: ");
    for (int i = 0; i < array_size; i++) {
      printf("%d ", array[i]);
    }
    printf("\n");

    free(array);
    return 0;
  }
```

Вывод

Программа реализует сортировку массива методом четно-нечетной сортировки Бетчера с использованием потоков для параллельной обработки. Пользователь задает массив и количество потоков, что позволяет распределить данные для сортировки между ними. Итоговый массив объединяется и дополнительно сортируется. Применение многопоточности в целом может дать существенный прирост к скорости и позволяет дополнительно оптимизировать свои программы.