Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу

«Операционные системы»

Управление потоками в ОС. Обеспечение синхронизации между потоками.

Студент: Попов Илья Павлович

Группа: М80-206Б-20

Вариант: 4

Преподаватель: Соколов Андрей Алексеевич

Дата: 27.11.2021

Оценка: 5

Подпись: _____

Москва, 2021

Постановка задачи

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

Вариант 4. Отсортировать массив целых чисел при помощи TimSort

Листинг программы

main.c

```
/*

Попов Илья М80-206Б-20

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix).

Ограничение потоков должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

В отчете привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков.

Получившиеся результаты необходимо объяснить.

Вариант №4

Отсортировать массив целых чисел при помощи TimSort

*/




#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include <pthread.h>

#include <math.h>


const int RUN = 4;
```

```
//Структура для перредачи данных функции insertionSort
typedef struct{
        int left;
        int right;
        int *mass;
}value;

pthread_mutex_t mutex;

// Обычная сортировка вставками
void* insertionSort(void* data) {
        value* res = (value*)data;

   for (int i = res->left + 1; i <= res->right; i++){
        pthread_mutex_lock(&mutex);
      int temp = res->mass[i];
      int j = i - 1;
      while (j >= res->left && res->mass[j] > temp){
         res->mass[j+1] = res->mass[j];
         j--;
      }
      res->mass[j+1] = temp;
      pthread_mutex_unlock(&mutex);//
   }

   pthread_exit(0);
   return NULL;
}

// Функция для слияния двух уже отсортированных RUN-ов
void merge(int arr[], int l, int m, int r){

   // Исходный массив разбиваем на две части - левый и правый массив
   int len1 = m - l + 1, len2 = r - m;
   int left[len1], right[len2];
   for (int i = 0; i < len1; i++)
      left[i] = arr[l + i];
```

```
    for (int i = 0; i < len2; i++)
        right[i] = arr[m + 1 + i];

    int i = 0;
    int j = 0;
    int k = l;

    // После сравнения мы объединяем эти два массива в более крупный подмассив
    while (i < len1 && j < len2) {
        if (left[i] <= right[j]) {
            arr[k] = left[i];
            i++;
        }
        else {
            arr[k] = right[j];
            j++;
        }
        k++;
    }

    // Копируем оставшиеся элементы слева, если они есть
    while (i < len1) {
        arr[k] = left[i];
        k++;
        i++;
    }

    // Копируем оставшиеся элементы справа, если они есть
    while (j < len2) {
        arr[k] = right[j];
        k++;
        j++;
    }
}

int min(int a, int b){
            if (a < b){return a;}
```

```
        else return b;
}


void timSort(int arr[], int n, int n_o_t) {

        pthread_mutex_init(&mutex, NULL);

        int num_of_treads = n / RUN + 1; // Вычисляем кол-во потоков
        printf("Кол-во необходимых потоков %d\n", num_of_treads);

        // Создаем массив сруктур, размер которого будет равен ко-ву потоков
        value treads_arr[num_of_treads];
        int c = 0;

   // Заполняем поля структуры, которую в последующем передадим в insertionSort
   for (int i = 0; i < n; i+=RUN){

                treads_arr[c].mass = arr;
                treads_arr[c].left = i;
                treads_arr[c].right = min((i+RUN-1), n-1);

        c++;
        }

        // Создаем массив идентификаторов потоков
        pthread_t threads[num_of_treads];


        if (n_o_t > num_of_treads){//если ограничение потоков больше, чем надо, ограничение будет тем, сколько надо =)
                n_o_t = num_of_treads;
        }
        printf("Кол-во потоков ограничено числом %d\n\n", n_o_t);
        while (num_of_treads > 0){
                // Создаем поток по идентификатору threads[i] и функции потока insertionSort и передаем потоку указатель на данные treads_arr
                for (int i = 0; i < n_o_t; i++) {
```

```
                pthread_create(&threads[i], NULL, insertionSort, &treads_arr[i]);
                printf("Запуск потока №%d\n", i);
            }
            // Ждем завершения потоков
            for (int i = 0; i < n_o_t; i++) {
                pthread_join(threads[i], NULL);
                    printf("Ожидание потока № %d\n", i);
            }
            num_of_treads -= n_o_t;
        }



    pthread_mutex_destroy(&mutex);


    // merge отсортированных RUN-ов
    for (int size = RUN; size < n; size = 2*size){

        for (int left = 0; left < n; left += 2*size) {

            int mid = left + size - 1;
            int right = min((left + 2*size - 1), (n-1));


            if(mid < right){
              merge(arr, left, mid, right);
            }
        }
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d  ", arr[i]);
    printf("\n\n");
```

```
}

void usage(){
	printf("Usage: ./a.out <ограничение по кол-ву потоков>\n\n");
}

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

	int array[] = {-2, 7, 15, -14, 0, 15, 0, 7, -7, -4, -13, 5, 8, -14, 12};
   int n = sizeof(array)/sizeof(array[0]);



   if(argc > 2){
		printf("Задайте ограничение по кол-ву потоков!\n");
		usage();
		return -1;
	}

	int n_o_t = atoi(argv[1]);

	if (n_o_t <= 0){
		printf("Некорректно задано ограничение по кол-ву потоков!\n");
		usage();
		return -2;
	}

   printf("Исходный массив:\n");
   printArray(array, n);

   timSort(array, n, n_o_t);

   printf("\n\nОтсортированный массив:\n");
   printArray(array, n);

   return 0;
}
```

Примеры работы

Тест № 1

```
lunidep@lunidep-VirtualBox:~/Desktop/OS_lab2/my$ gcc -g -Wall main.c -pthread -lrt -std=c99
lunidep@lunidep-VirtualBox:~/Desktop/OS_lab2/my$ ./a.out 3
Given Array is
-2  7  15  -14  0  15  0  7  -7  -4  -13  5  8  -14  12

Кол-во необходимых потоков 4
Кол-во потоков ограничено числом 3

Запуск потока №0
Запуск потока №1
Запуск потока №2
Ожидание потока № 0
Ожидание потока № 1
Ожидание потока № 2
Запуск потока №0
Запуск потока №1
Запуск потока №2
Ожидание потока № 0
Ожидание потока № 1
Ожидание потока № 2

After Sorting Array is
-14  -13  -7  -4  -2  0  0  5  7  7  8  -14  12  15  15
```

Тест № 2

```
lunidep@lunidep-VirtualBox:~/Desktop/OS_lab2/my$ gcc -g -Wall main.c -pthread -lrt -std=c99
lunidep@lunidep-VirtualBox:~/Desktop/OS_lab2/my$ ./a.out 9
Given Array is
-2  7  15  -14  0  15  0  7  -7  -4  -13  5  8  -14  12

Кол-во необходимых потоков 4
Кол-во потоков ограничено числом 4
```

Запуск потока №0

Запуск потока №1

Запуск потока №2

Запуск потока №3

Ожидание потока № 0

Ожидание потока № 1

Ожидание потока № 2

Ожидание потока № 3

After Sorting Array is

-14 -14 -13 -7 -4 -2 0 0 5 7 7 8 12 15 15

Тест № 3

lunidep@lunidep-VirtualBox:~/Desktop/OS_lab2/my$ gcc -g -Wall main.c -pthread -lrt -std=c99

lunidep@lunidep-VirtualBox:~/Desktop/OS_lab2/my$ ./a.out

Задайте ограничение по кол-ву потоков!

Usage: ./a.out <ограничение по кол-ву потоков>

lunidep@lunidep-VirtualBox:~/Desktop/OS_lab2/my$ ./a.out -3

Некорректно задано ограничение по кол-ву потоков!

Usage: ./a.out <ограничение по кол-ву потоков>

Вывод

В ходе данной лабораторной работы я познакомился с многопоточностью в программировании, которая, несомненно, является важным механизмом в наше время. В семействах ОС Windows - каждая программа запускает один процесс выполнения, в котором находится как минимум один поток. В процессе может находиться множество потоков, между которыми делится процессорное время. Один процесс не может напрямую обратиться к памяти другого процесса, а потоки же разделяют одно адресное пространство одного процесса. То есть в Windows процесс - это совокупность потоков.

Также стоит отметить про межпоточные конфликты. Чтобы защитить память, с которой работает один процесс от другого процесса, необходимо пользоваться mutex'ами. Они блокируют доступ к ресурсам для всех потоков кроме того, который владеет мьютексом. Он же единственный, кто может его разблокировать этот доступ.