Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 3
по курсу «Операционные системы»

Студент:	Чурсина Н. А.
Группа:	М8О-208Б-18
Вариант:	3
Преподаватель:	Миронов Е. С.
Оценка:	
Дата:	

Москва, 2019

1. Постановка задачи

Отсортировать массив строк при помощи параллельной сортировки слиянием.
Операционная система: MacOS.

Целью лабораторной работы является:

Приобретение практических навыков в:

- Управление процессами в ОС
- Обеспечение обмена данных между процессами посредством каналов

Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска вашей программы, или алгоритмом.

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

2. Решение задачи

pthread_t threads[THREAD_MAX] - создаем массив идентификаторов потока
pthread_create(&threads[i], NULL, merge_sort_tread, NULL); - создаем потоки

Используемые системные вызовы:

1. **pthread_join —** дожидается завершения переданного потока, после чего получает

его выходное значение и позволяет программе продолжить работу.

int pthread_join(pthread_t thread, void **value_ptr);

Откладывает выполнение вызывающего (эту функцию) потока, до тех пор, пока не будет выполнен поток thread. Когда pthread_join выполнилась успешно, то она возвращает 0. Если поток явно вернул значение (это то самое значение SUCCESS, из нашей функции), то оно будет помещено в переменную value_ptr. Возможные ошибки, которые возвращает pthread_join

2. **pthread_create** — создает новый поток выполнения в программе.

int pthread_create(*ptherad_t, const pthread_attr_t *attr, void* (*start_routine)(void*), void *arg);
Функция получает в качестве аргументов указатель на поток, переменную типа pthread_t, в которую, в случае удачного завершения сохраняет id потока. pthread_attr_t – атрибуты потока. В случае если используются атрибуты по умолчанию, то можно передавать NULL. start_routin – это непосредственно та функция, которая будет выполняться в новом потоке. arg – это аргументы, которые будут переданы функции.

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ:

Входной массив: 129 151 255 238 177 160 217 151 55 90 0 218 13 7 122 85 27 113 173 153 105 109 76 164 117 167 13 69 148 198 239 224 52 113 111 16 233 200 39 210 106 9 171 99 139 163 164 124 83 145 102 244 216 198 79 143 61 143 219 13 238 209 97 189 105 3 182 104 102 170 200 81 83 27 173 182 75 205 50 92 141 240 58 96 167 91 233 104 6 44 227 195 209 189 219 14 45 84 23 120 28 7 53 217 80 247 110 140 252 43 115 3 199 125 248 78 243 202 169 255 15 226 211 30 212 195 174 34 126 128 84 198 4 213 151 165 73 114 214 25 139 10 150 61 242 95 156 71 45 110 204 52 215 200 227 122 120 109 17 165 96 64 116 150 44 205 4 69 116 154 239 138 243 32 49 226 115 212 251 205 52 203 243 238 167 172 171 10 192 7 225 112 109 37 113 235 67 254 144 243 8 245 65 239 246 56 149 182 99 105 238 9 120 68 87 42 63 25 195 33 133 108 186 71 95 134 244 9 159 32 245 66 173 202 226 73 255 81 192 187 124 85 34 68 251 178 174 103 37 188 235 221 17 88 161 75

part- 0

part- 1

part- 2

part- 3

part- 3

part- 4

part- 5

part- 5

part- 7

part- 8

part- 9

part- 11

part- 11

part- 12

part- 13

part- 14

Выходной массив: 0

3

3

4

4

6

7

7

7

8

9

9

9

10

10

13

13

13

14

15

16

17

17

23

25

25

27

27

28

30

32

32

33

34

34

37

37

39

42

43

44

44

45

45

49

50

52

52

52

53

55

56

58

61

61

63

64

65

66

67

68

68

69

69

71

71

73

73

75

75

76

78

79

80

81

81

83

83

84

84

85

85

87

88

90

91

92

95

95

96

96

97

99

99

102

102

103

104

104

105

105

105

106

108

109

109

109

110

110

111

112
113
113
113
114
115
115
116
116
117
120
120
120
122
122
124
124
125
126
128
129
133
134
138
139
139
140
141
143
143
144
145
148
149
150
150
151
151
151
153
154
156
159
160

204

205

205

205

209

209

210

211

212

212

213

214

215

216

217

217

218

219

219

221

224

225

226

226

226

226

227

227

233

233

235

235

238

238

238

238

239

239

239

240

242

243

243

243

243

244

244

245

245

246

247

248

251

251

252

254

255

255

255

Количество потоков: 16%

3. Руководство по использованию программы

Компиляция и запуск программного кода в *MacOs Mojave* :

gcc -o main main.c

./main

3. Листинг программы

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
/*
 При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска вашей программы, или алгоритмом.
 3.    Отсортировать массив строк при помощи параллельной сортировки слиянием.

 */

int MAX = 256;  // Количество элементов в массиве
int THREAD_MAX; //количество потоков

int part=0; //переменная, которая будет хранить количество потоков
int* a ;    //указатель на массив для хранения элементов



// функция для слияния двух частей
void merge(int low, int mid, int high)
{
    int n1 = mid - low + 1, nr = high - mid, i, j;

    int* left = calloc(n1,sizeof(int));//выделение памяти под левую половину
    int* right = calloc(nr,sizeof(int));//выделение памяти под правую половину

    for(i = 0; i < n1; i++)//заполняем левую половину
        left[i] = a[i + low];

    for(i = 0; i < nr; i++)// заполняем правую половину
        right[i] = a[i + mid + 1];
```

```
    int k = low;
    i = j = 0;

    // объединяем левую и правую половины в порядке возрастания
    while(i < n1 && j < nr)
    {
        if(left[i] <= right[j])
            a[k++] = left[i++];
        else
            a[k++] = right[j++];
    }

    // вставляем оставшиеся значения из левой половины
    while(i < n1) {
        a[k++] = left[i++];
    }

    // вставляем оставшиеся значения из правой половины
    while(j < nr) {
        a[k++] = right[j++];
    }



}

// функция сортировки слиянием
void merge_sort(int low, int high)
{
    // вычисление средней точки массива
    int mid = low + ((high - low) / 2);
    if(low < high) {

        //сортировка первой половины
        merge_sort(low, mid);

        // сортировка второй половины
        merge_sort(mid + 1, high);

        // объединение половинок
        merge(low, mid, high);
    }
}

//функция сортировки с использованием потоков
void* merge_sort_tread()
{
    //получает идентификатор потока
    int thread_part = part;
    part+=1;

    // вычисляем нижнюю и верхнюю границу массива
    int low = thread_part * (MAX / THREAD_MAX);
    int high = (thread_part + 1) * (MAX / THREAD_MAX) - 1;

    //printf("+%d %d %d\n",part,low,high);

    //   обновление средней точки
    int mid = low + (high - low) / 2;


    if(low < high)
    {
        merge_sort(low, mid);
        merge_sort(mid + 1, high);
        merge(low, mid, high);
    }


}



void marge_rec(int tread_m)//рекурсивная функция сбора данных после отработки потоков
{
    if(tread_m>THREAD_MAX)
        return;
```

```
    marge_rec(tread_m*2);

    int minsize = MAX/tread_m;

    for(int i=0;i<tread_m;i++)
    {

        int low = i * minsize;
        int high = (i + 1) * (minsize) - 1;
        int mid = low + (high - low) / 2;

        // printf("$%d %d\n",low,high);
        // обновление средней точки

        merge(low, mid, high);



    }

}
// главная функция
int main()
{

    //находим максимальное количество потоков
    int pow_ = 0;
    while (1)
    {
        int p = MAX >> pow_;
        if(p==0||p==1)
            break;

        pow_++;
    }

    MAX=pow(2,pow_);
    THREAD_MAX = MAX/16;


    a = calloc(MAX,sizeof(int));//выделение памяти под левую половину
    printf("Входной массив: ");
    //----------------------генерация массива чисел
    int i;
    srand(time(NULL)); // устанавливаем начальное значение генератора случайных чисел
    for( i=0; i < MAX; i++)
    {
        a[i] = rand() % MAX;
        printf("%d ",a[i]);
    }
    printf("\n");


    pthread_t threads[THREAD_MAX];//создаем массив идентификаторов потока


    int status;

    // создаем потоки
    for(int i = 0; i < THREAD_MAX; i++)
    {
        printf("part- %d\n",part);

        status = pthread_create(&threads[i], NULL, merge_sort_tread, NULL);
        if (status != 0)
            printf("main error: can't create thread, status = %d\n", status);


    }

    //ожидание завершения всех потоков
    for(int i = 0; i < THREAD_MAX; i++)
        pthread_join(threads[i], NULL);


    //объединяем результаты работы потоков
```

```
    marge_rec(2);
    merge(0, (MAX - 1)/2, MAX - 1);




    // вывод отсортированного массива
    printf("Выходной массив: ");
    for(int i = 0; i < MAX; i++)
        printf("%d\n",a[i]);

    printf("Количество потоков: %d",THREAD_MAX);
    return 0;
}
```

5. Вывод

Современные программы редко работают в одном процессе или потоке. Довольно частая ситуация: нам необходимо запустить какую-то программу из нашей. Также многие программы создают параллельные потоки.

При работе с потоками и процессами необходимо быть весьма осторожным, так как возможно допустить различные ошибки, которые затем будет сложно отловить и исправить.