Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Операционные системы»

Студент: Э. Л. Носов
Преподаватель: Е. С. Миронов
Группа: М8О-307Б
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №3

Цель работы: Целью является приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

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

Вариант №2: Отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки

1 Описание

Алгоритм быстрой сортировки подразумевает разделение массива на две части относительно опорного элемента, после чего над каждым из подмассивов проделывается тоже самое. В случае параллельности каждому процессу удобно отдавать получившийся в результате разделения массива подмассив дочернему процессу, для контроля количества процессов каждый "родительделится"со своим "ребенком"частью своих процессов. Если процесс не может создать дочерний процесс, то он продолжает обработку своей части массива линейно (линейный quicksort реализован и процесс обращается к нему).

2 Исходный код

Функция quick_sort_parallel подготавливает данные для функции quick_sort_parallel_thread, после чего эта функция сначала запускается главным потоком, а затем передается в функцию pthread_create при создании нового процесса. Эта функция вызывает функцию partition, которая нужным образом обрабатывает фрагмент массива, затем проверяет, можно ли создать поток, если можно, то формирует данные для него, если нельзя, то продолжает обработку своего фрагмента линейно.

lab3.c

```
#include"stdio.h"
#include"pthread.h"
#include"stdlib.h"
#include<time.h>
#include<unistd.h>

typedef struct{
    int *array, left, right, thread_number, isthread;
} pthrData;

int partition(int* array, int left, int right){
    int pivot = array[(left+right)/2], i = left, j = right, tmp;
    while(1){
        while(array[i]<pivot) i++;
        while(array[j]>pivot) j--;
        if(i>=j) return j;
        tmp = array[i];
        array[i] = array[j];
        array[j] = tmp;
        i++;
        j--;
    }
    return j;
}

void quick_sort(int*array, int left, int right){
    if(left<right){
        int p=partition(array, left, right);
        quick_sort(array, left, p);
        quick_sort(array, p+1, right);
    }
}

void* quick_sort_parallel_thread(void* thread_data/* int*array, int left, int right */){
    pthrData *data=(pthrData*)thread_data;
    if(data->left<data->right){
        int p=partition(data->array, data->left, data->right), tmp;
        pthrData* dataToPthr=NULL;
        pthread_t thread;
        if(data->thread_number > 0){
            data->thread_number--;
            dataToPthr=(pthrData*)malloc(sizeof(pthrData));
            dataToPthr->array=data->array;
            dataToPthr->right=data->right;
            data->right=p;
            dataToPthr->left=p+1;
            tmp=data->thread_number;
            data->thread_number=tmp/2;
            dataToPthr->thread_number=tmp-data->thread_number;
            dataToPthr->isthread = 1;
            pthread_create(&thread, NULL, quick_sort_parallel_thread, dataToPthr);
            quick_sort_parallel_thread(data);
```

```
53              pthread_join(thread, NULL);
54              free(dataToPthr);
55          }
56          else{
57              quick_sort(data->array, data->left, data->right);
58          }
59          return 0;
60      }
61  }
62
63
64  void quick_sort_parallel(int *array, int left, int right, int thread_number){
65      pthrData data;
66      data.array=array;
67      data.left=left;
68      data.right=right;
69      data.thread_number=thread_number;
70      data.isthread=0;
71      quick_sort_parallel_thread(&data);
72  }
73
74  int main(int argc, char *argv[]){
75      int *array, *array1, length = 0, capacity = 1;
76      array=(int*)malloc(sizeof(int));
77      while(scanf("%d", &array[length]) > 0){
78          length++;
79          if(capacity == length){
80              capacity *= 2;
81              array = (int*)realloc(array, sizeof(int)*capacity);
82          }
83      }
84      array = (int*)realloc(array, sizeof(int)*length);
85      array1 = (int*)malloc(sizeof(int)*length);
86      for(int i=0; i<length; i++){
87          array1[i]=array[i];
88      }
89      if(argc == 1){
90          printf("\nquick\n");
91          quick_sort(array, 0, length-1);
92      }
93      else{
94          printf("\nparallel\n");
95          quick_sort_parallel(array, 0, length-1, (int)strtol(argv[1], NULL, 10));
96      }
97      for(int i = 0; i < length; i++){
98          printf("%d ", array[i]);
99      }
100     free(array1);
101     free(array);
102 }
```

3 Консоль

```
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./lab 3
6 4 8 0 7 6 3 1 5 5 3 4 2 1 5
parallel
0 1 1 2 3 3 4 4 5 5 5 6 6 7 8 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./l
6 4 8 0 7 6 3 1 5 5 3 4 2 1 5
quick
0 1 1 2 3 3 4 4 5 5 5 6 6 7 8
```

4 Тест производительности

Сравнивать скорость параллельного алгоритма я буду со скоростью линейного алгоритма собственной реализации. Для этого я несколько изменил программу, сделав две: одну для вывода времени работы параллельного алгоритма и одну для вывода времени работы линейного.

```
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ python3 generator.py 3000 > test

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 3 < test

 0.000511

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./quick < test

 0.000239

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ python3 generator.py 300000 > te

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 3 < test

 0.019995

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./quick < test

 0.032713

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ python3 generator.py 3000000 > t

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 3 < test

 0.255050

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./quick < test

 0.339074

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 3 < test

 0.248117

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 5 < test

 0.144632

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 7 < test

 0.140288

 maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 9 < test
```

```
0.095905

maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 10 < test

0.099335

maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 11 < test

0.095817

maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 13 < test

0.093104

maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 13 < test

0.094859

maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 13 < test

0.091846

maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab3$ ./par 17 < test

0.092581
```

На маленьких тестах параллельный алгоритм проигрывает в скорости, что можно связать с системными вызовами, нужными для создания потоков, но на более объемных тестах параллельный алгоритм заметно выигрывает, при этом при увеличении потоков время выполнения значительно уменьшается вплоть до 9 потоков, после чего меняется незначительно, что связано с тем, что на моем устройстве максимальное доступное колличество потоков равно 12.

5 Выводы

Выполнив лабораторную работу №3 по курсу «Операционные системы», я освоил управление потоками с помощью библиотеки pthread и смог с приемлемой эффективностью реализовать параллельный алгоритм быстрой сортировки.