Московский авиационный институт
(национальный исследовательский университет)
Институт № 8 «Информационные технологии и прикладная математика»

Лабораторная работа №3
по курсу «Операционные системы»

Студент: Минеева Светлана Алексеевна

Группа: М8О-210Б-21

Преподаватель: Миронов Е.С

Вариант №3
Оценка: ____
Дата: 17.10.2022
Подпись: _____

Москва, 2022

Содержание:

1. Цель работы

Приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

2. 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.

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.

3. Вариант задания

Вариант №3: отсортировать массив целых чисел при помощи параллельной сортировки слиянием.

4. Общие сведения о программе

Программа компилируется из файла lab3.c. Кроме стандартной библиотеки в программе также используются следующие библиотеки: pthread.h, time.h, stdlib.h. Функции, использующиеся в программе для работы с потоками:

1) pthread_create - создание потока;
2) pthread_join – ожидание результата от потока.

5. Алгоритм решения

Благодаря тому, что сортировка слиянием построена на принципе «разделяй и властвуй», выполнение данного алгоритма можно эффективно распараллелить.

Этапы сортировки слиянием:

1) Массив рекурсивно разделяется пополам до тех пор, пока размер очередного подмассива не станет равным одному;
2) Каждая половина сортируется отдельно;
3) Два массива сливаются в один.

Так как сортировка каждого разделения массива не зависит от других подмассивов, её можно распараллелить и выполнять сортировки подмассивов разными потоками.

6. Текст программы

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

int *array;

struct task {
  int number;
  int left;
  int right;
};

void merge(int low, int mid, int high) {
  int *left_array = malloc((mid - low + 1) * sizeof(int));
  int *right_array = malloc((high - mid) * sizeof(int));

  for(int i = 0; i < mid - low + 1; i++) {
    left_array[i] = array[i + low];
  }

  for(int i = 0; i < high - mid; i++) {
    right_array[i] = array[i + mid + 1];
  }

  int count = low;
  int i = 0, j = 0;

  while(i < mid - low + 1 && j < high - mid) {
    if(left_array[i] <= right_array[j]) {
      array[count] = left_array[i];
```

```
      count += 1;
      i += 1;
    }
    else {
      array[count] = right_array[j];
      count += 1;
      j += 1;
    }
  }

  while(i < mid - low + 1) {
    array[count] = left_array[i];
    count += 1;
    i += 1;
  }

  while(j < high - mid) {
    array[count] = right_array[j];
    count += 1;
    j += 1;
  }

  free(left_array);
  free(right_array);
}

void merge_sort(int low, int high) {
  int mid = (low + high) / 2;

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

void *merge_sort_thread(void *arg) {
  struct task *tsk = arg;
  int mid = (tsk -> left + tsk -> right) / 2;

  if(tsk -> left < tsk -> right) {
    merge_sort(tsk -> left, mid);
    merge_sort(mid + 1, tsk -> right);
    merge(tsk -> left, mid, tsk -> right);
  }
}

int main(int argv, char *argc[]) {
  struct task *control_param;
  int size, number_threads;
  printf("Number elements: \n");
  scanf("%d", &size);
  number_threads = *argc[1];
```

```
  array = malloc(sizeof(int) * size);
  printf("Array of integers: \n");

  for(int i = 0; i < size; i++) {
    scanf("%d", &array[i]);
  }

  pthread_t threads[number_threads];
  struct task tsklist[number_threads];

  if(number_threads > size) {
    number_threads = size;
  }

  int length = size / number_threads;
  int low = 0;

  for(int i = 0; i < number_threads; i++, low += length) {
    control_param = &tsklist[i];
    control_param -> number = i;
    control_param -> left = low;
    control_param -> right = low + length - 1;

    if(i == (number_threads - 1)) {
      control_param -> right = size - 1;
    }
  }

  for(int i = 0; i < number_threads; i++) {
    control_param = &tsklist[i];
    pthread_create(&threads[i], NULL, merge_sort_thread, control_param);
  }

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

  struct tsk *tskm = &tsklist[0];

  for(int i = 1; i < number_threads; i++) {
    struct task *tsk = &tsklist[i];
    merge(tskm -> left, tsk -> left - 1, tsk -> right);
  }

  printf("Sorted array: ");

  for(int i = 0; i < size; i++) {
    printf("%d ", array[i]);
  }

  printf("\n");
  return 0;
}
```

7. Демонстрация работы программы

Last login: Sun Oct 16 22:04:15 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
MacBook-Pro-MacBook:~ macbookpro$ ./lab3 4
Number elements:
4
Array of integers:
6 0 7 8
Sorted array: 0 6 7 8
MacBook-Pro-MacBook:~ macbookpro$ ./lab3 2
Number elements:
4
Array of integers:
4 3 2 1
Sorted array: 1 2 3 4
MacBook-Pro-MacBook:~ macbookpro$ ./lab3 2
Number elements:
2
Array of integers:
0 0
Sorted array: 0 0
MacBook-Pro-MacBook:~ macbookpro$ ./lab3 1
Number elements:
1
Array of integers:
4
Sorted array: 4
MacBook-Pro-MacBook:~ macbookpro$ ./lab3 2
Number elements:
10
Array of integers:
10 -1 3 7 2 -90 7 0 8 1
Sorted array: -90 -1 0 1 2 3 7 7 8 10
MacBook-Pro-MacBook:~ macbookpro$

8. Исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков

Метрики параллельных вычислений - это система показателей, позволяющая оценивать преимущества, получаемые при параллельном решении задачи на

n процессорах, по сравнению с последовательным решением той же задачи на единственном процессоре. С другой стороны, они позволяют судить об обоснованности применения данного числа процессоров для решения конкретной задачи.

Базисом для определения метрик являются следующие характеристики вычислений:

Tp – время выполнения на p различных потоках/вычислительных ядрах Sp – ускорение. Sp = T1/Tp, Sp < p
Xp – эффективность/загруженность. Xp = Sp/p, Xp<1

Приведу таблицу с результатами вычисления метрик для размера массива 100:

p	T1 (sec)	Tp (sec)	Ускорение (Sp=T1/Tp)	Эффективность (Xp=Sp/p)
4	2.889	2.871	1.01	0.252
6	2.889	2.191	1.32	0.22
8	2.889	1.950	1.48	0.185
10	2.889	1.808	1.598	0.1598

Приведу таблицу с результатами вычисления метрик для размера массива 255:

p	T1 (sec)	Tp (sec)	Ускорение (Sp=T1/Tp)	Эффективность (Xp=Sp/p)
4	5.020	3.565	1.40	0.35
6	5.020	3.540	1.42	0.237
8	5.020	2.911	1.72	0.215
10	5.020	2.740	1.83	0.183

Итак, алгоритм лучше параллелится при больших количествах входных данных и небольшого числа потоков. В работе решета потоки могут выполнять лишнюю работу, а также тратится дополнительное время на создание и ожидание этих потоков. Чем больше потоков, тем больше лишней работы будет выполняться потоками.

9. Вывод

Я составила программу на языке Си, обрабатывающую данные в многопоточном режиме. В процессе выполнения данной лабораторной работы я приобрела практические навыки в управлении потоками ОС и обеспечении синхронизации между потоками. Мною были освоены функции для работы с потоками, такие как pthread_create(), pthread_join().

В ходе этой лабораторной работы я отметила, что создание потоков происходит быстрее, чем создание процессов, и все потоки используют одну и ту же область данных, поэтому многопоточность – один из способов ускорить обработку каких-либо данных: выполнение однотипных, не зависящих друг от друга задач, можно поручить отдельным потокам, которые будут работать параллельно.