Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №X по курсу
«Операционные системы»

Студент: Друхольский А.К.
Группа: М8О-207Б-21
Вариант: 17
Преподаватель: Черемисинов Максим
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/ssForz/OS-labs

Постановка задачи

Цель работы

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. 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 17) Найти в большом целочисленном массиве минимальный элемент

Общие сведения о программе

Программа компилируется из файла main.c при помощи cmake. В программе реализована многопоточность. Функции для работы с потоками. которые я использовал:

- pthread_create() — создание потока с передачей ему аргументов. В случае успеха возвращает 0.
- `pthread_join()` — ожидает завершения потока обозначенного `THREAD_ID`. Если этот поток к тому времени был уже завершен, то функция немедленно возвращает значение.
- pthread_mutex_init() — инициализация мьютекса
- pthread_mutex_lock() — блокировка мьютекса
- pthread_mutex_lock() — открытие доступа к мьютексу
- pthread_mutex_destroy() - удаление мьютекса

Общий метод и алгоритм решения

Требуется найти минимальный элемент в массиве. Суть будет в том, что массив разобьётся на количество частей, равное количеству потоков. В каждом потоке будет искаться минимальный элемент подмассива и сравниваться с глобальным минимальным элементом (важно заметить, что мы используем mutex, чтобы избежать несвоевременного изменения глобальной переменной, содержащей минимальное значение). Количество потоков логично будет ограничить количеством элементов в массиве.

Так же в функции, которую выполняет поток добавим вывод надписи, что сейчас процедура выполняется в данном потоке. Сделаем это для наглядности параллельности процедур.

Исходный код

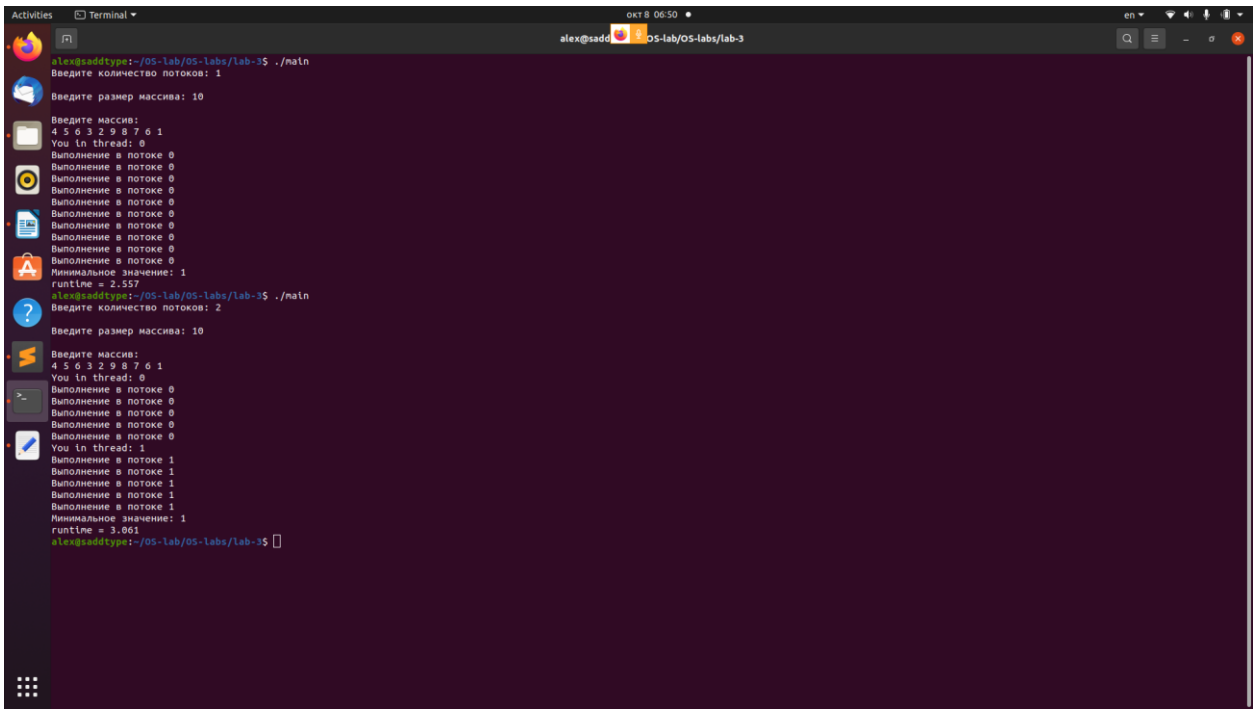
```
#include<pthread.h>
#include<iostream>
#include <ctime>

using namespace std;

static int minimal = 1e9;
pthread_mutex_t mutex;
int flag = 0;


struct arg_to_thread {
   int* big_array;
   int partition;
   int num_of_thread;
   int count_threads;
   int size_array;
};

void* thread_func(void *args)
{
   arg_to_thread* arguments = (arg_to_thread*) args;
   int num_of_thread = arguments->num_of_thread;
   int partition = arguments->partition;
```

```
    flag = 1;
    int count_threads = arguments->count_threads;
    int size_array = arguments->size_array;
    int* big_array = arguments->big_array;
      cout<<"You in thread: "<<num_of_thread<<endl;

      if (num_of_thread != count_threads - 1) {
        for (int j = num_of_thread * partition; j < num_of_thread * partition + partition; ++j) {
          cout<<"Выполнение в потоке "<<num_of_thread<<endl;
          pthread_mutex_lock(&mutex);
            if (big_array[j] < minimal) {
              minimal = big_array[j];
            }
          pthread_mutex_unlock(&mutex);
        }
      } else {
        for (int j = size_array - 1; j > size_array - partition - 1; --j) {
          cout<<"Выполнение в потоке "<<num_of_thread<<endl;
          pthread_mutex_lock(&mutex);
            if (big_array[j] < minimal) {
              minimal = big_array[j];
            }
          pthread_mutex_unlock(&mutex);
        }

      }



    return 0;
}

int main(int argc, char const *argv[])
{
    srand(time(0));
    int count_threads;
```

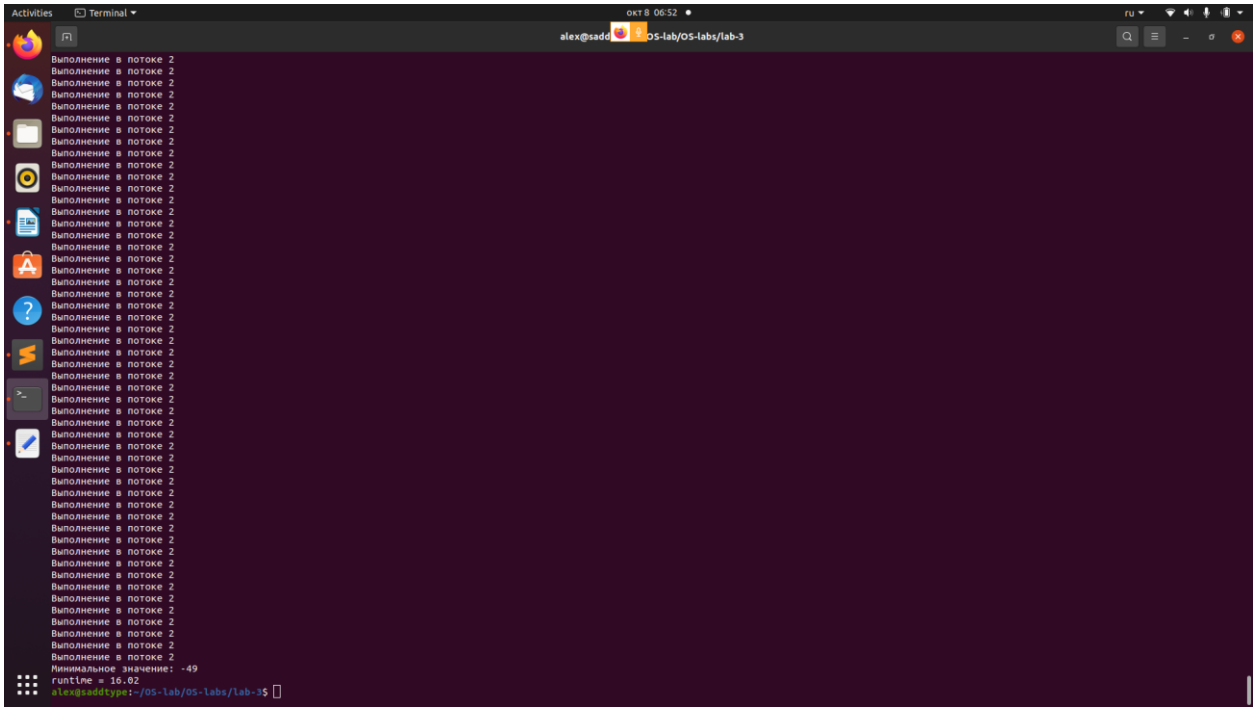
```
cout<<"Введите количество потоков: ";
cin>>count_threads;
cout<<endl;
int size_array;
cout<<"Введите размер массива: ";
cin>>size_array;
cout<<endl;
int big_array[size_array];
cout<<"Введите массив: "<<endl;
for(int i = 0; i < size_array; ++i) {
   cin>>big_array[i];
}
if (count_threads > size_array) {
   cout<<"Количество потоков больше максимального"<<endl;
   count_threads = size_array;
}
pthread_t threads[count_threads];

pthread_mutex_init(&mutex, NULL);
int partition = size_array / count_threads;
struct arg_to_thread arg;
arg.partition = partition;
arg.big_array = big_array;
arg.count_threads = count_threads;
arg.size_array = size_array;

for (int i = 0; i < count_threads; i++) {
   arg.num_of_thread = i;
   if (i == count_threads - 1) {
      partition += size_array % count_threads;
      arg.partition = partition;
   }
   int status = pthread_create(&threads[i], NULL, thread_func, (void*)&arg);
   while(flag != 1) {
      //ожидание пока функции передадутся аргументы
   }
```

```
        flag = 0;
        if (status != 0) {
            cout<<"Create thread error"<<endl;
        }
    }

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

    pthread_mutex_destroy(&mutex);

    cout<<"Минимальное значение: "<<minimal<<endl;
    cout << "runtime = " << clock()/1000.0 << endl;
    return 0;

}
```

Демонстрация работы программы



Демонстрация работы программы для массива из 10 чисел. 1 и 2 потока.



Пример с большим массивом (1000 чисел). 4 потока (вывод программы не помещается полностью, поэтому показан только результат).

Выводы

Для задачи нахождения минимального элемента многопоточность (которая реализована моим способом) не ускоряет выполнение программы. При больших значениях, когда размер массива сильно больше количества потоков, время выполнения программы на нескольких потоках может быть в 3-4 раза больше чем при выполнении на одном потоке. Скорее всего это связано с тем, что задача нахождения минимального элемента в массиве имеет линейную сложность. Поэтому разбиение массива на более маленькие и распределение их по потокам не ускоряет алгоритм, а только замедляет (незначительно, при малых размерах массива). Это происходит, потому что мы в каждом потоке обращаемся к одной и той же памяти (в нашем случае это переменная minimal) и ждём доступа.

Данная лабораторная работа познакомила меня с потоками, научила работать с многопоточностью и защищать участки памяти с помощью mutex.