

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: Друхольский А.К.
Группа: М8О-207Б-21
Вариант: 17
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

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 обработке 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 make. 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_unlock()` — 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 (изначально была попытка сравнивать минимум каждого потока с глобальной переменной, используя при этом мьютекс, но это сказалось отрицательно на эффективности, поэтому было принято решение выделить из каждого подмассива минимальный элемент отдельно). 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 (на этапе тестирования). Сделаем это для наглядности параллельности процедур.

Исходный код

```
#include<pthread.h>
#include<iostream>
#include <ctime>
#include<vector>
#include<fstream>
#include<chrono>

using namespace std;

static int minimal = 1e9;
pthread_mutex_t mutex;
int flag = 0;
vector<int> result(100);

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

void* thread_func(void *args)
{
    int minimum = 1e9;
    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;

if (num_of_thread != count_threads - 1) {
    for (int j = num_of_thread * partition; j < num_of_thread * partition + partition; ++j) {
        if (big_array[j] < minimum) {
            minimum = big_array[j];
            result[num_of_thread] = big_array[j];
        }
    }
} else {
    for (int j = size_array - 1; j > size_array - partition - 1; --j) {
        if (big_array[j] < minimum) {
            minimum = big_array[j];
            result[num_of_thread] = big_array[j];
        }
    }
}

return 0;
}

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

    ifstream file("./src/big_massives.txt");
    int count_threads;
    cout<<"Введите количество потоков: ";
    cin>>count_threads;
    cout<<endl;
    int size_array;
    cout<<"Введите размер массива: ";
    cin>>size_array;
    cout<<endl;
    vector<int>big_array(size_array);
    cout<<"Введите массив: "<<endl;
    for(int i = 0; i < size_array; ++i) {
        string s;
        getline(file, s);
    }
}

```

```

    big_array[i] = atoi(s.c_str());
}
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.data();
arg.count_threads = count_threads;
arg.size_array = size_array;

chrono::steady_clock::time_point begin = chrono::steady_clock::now();
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);
}
chrono::steady_clock::time_point end = chrono::steady_clock::now();

pthread_mutex_destroy(&mutex);
int minimum2 = 1e9;
for (int i = 0; i < count_threads; ++i) {
    if (result[i] < 1e9) {
        minimum2 = result[i];
    }
}

```

```

    }
    cout<<"Минимальное значение: "<<minimum2<<endl;
    cout<<chrono::duration_cast<chrono::microseconds>(end-begin).count();
    return 0;
}
//find_minimal(big_array, i, partition)

```

Демонстрация работы программы

Продemonстрируем работу многопоточного алгоритма на массиве из 10^8 чисел

```

alex@saddtype:~/OS-lab(for testing)/OS-labs/lab-3$ ./main
Введите количество потоков: 2

Введите размер массива: 100000000

Введите массив:
Минимальное значение: -49
115591alex@saddtype:~/OS-lab(for testing)/OS-labs/lab-3$ ./main
Введите количество потоков: 4

Введите размер массива: 100000000

Введите массив:
Минимальное значение: -49
50921alex@saddtype:~/OS-lab(for testing)/OS-labs/lab-3$ ./main
Введите количество потоков: 8

Введите размер массива: 100000000

Введите массив:
Минимальное значение: -49
alex@saddtype:~/OS-lab(for testing)/OS-labs/lab-3$ ./main
Введите количество потоков: 1

Введите размер массива: 100000000

Введите массив:
Минимальное значение: -49

```

Из тестирования видно, что значения при 1, 2, 4, 8, 16 потоках сильно отличаются. Особенно это разница заметна при малом количестве потоков (отличие почти в два раза при 1 и 2 потоках).

Проверим вывод утилиты strace (запишем в отдельный файл, а потом найдём вызовы создания потоков). Проверим на тесте с 4-мя потоками.

```
alex@asddtype: /OS-lab(for testing)/OS-lab-3$ cat log.txt |grep clone
0, child_stack=0x7faedd869f80, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARPID, parent_tid=[47850], tls=0x7faedd86a780
0, child_tidptr=0x7faedd86a9d0) = 47850
0, child_stack=0x7faedd868f80, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARPID, parent_tid=[47851], tls=0x7faedd869780
0, child_tidptr=0x7faedd8699d0) = 47851
0, child_stack=0x7faedc867f80, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARPID, parent_tid=[47852], tls=0x7faedc868780
0, child_tidptr=0x7faedc8699d0) = 47852
0, child_stack=0x7faedc866f80, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARPID, parent_tid=[47853], tls=0x7faedc867780
```

Можно сделать вывод, что создание потоков реализовано в Linux с помощью утилиты `clone()`. Отличие от `fork()` в более тонкой настройке. С помощью этой утилиты в целом можно самостоятельно создавать потоки.

Выводы

Я познакомился с многопоточностью и смог реализовать её для моей задачи. Так же мне удалось выявить зависимость между количеством потоков и временем выполнения. При малом количестве потоков (≤ 4) разницы во времени выполнения очень значительна.

Дальнейшее увеличение количества потоков приводит лишь к незначительному набору эффективности (много времени тратится на инициализацию потоков). Так же, с помощью утилиты strace, я узнал как реализовано создание потоков в линуксе.

Многопоточность – полезная вещь в различных проектах, а благодаря этой лабораторной работе я получил базовые навыки в работе с потоками.