Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу
«Операционные системы»

УПРАВЛЕНИЕ ПОТОКАМИ

Студент: Мирошников Дмитрий Евгеньевич

Группа: М8О-210Б-22

Вариант: 15

Преподаватель: Соколов Андрей Алексеевич

Оценка:_____

Дата:_____

Подпись:_____

Москва, 2023

Постановка задачи

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 максимального количества потоков, работающих в один момент времени, 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 входных данных и количества потоков. Получившиеся результаты необходимо объяснить.
Вариант 15 : Есть колода из 52 карт, рассчитать экспериментально (метод Монте-Карло) вероятность того, что сверху лежат две одинаковых карты. Количество раундов задаётся ключом программы.

Основные сведения о программе

В программа компилируется при помощи запуска исполняемого файла, ранее собранного с помощью cmake. В зависимости от выбранного файла варьируется число аргументов: в assync необходимо последовательно указать число раундов метода и число потоков, в то время как в sync необходимо указать только число раундов.
В своей программе я использую следующие системные вызовы
1) **pthread_create** – функция создает новый поток.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.

2) **pthread_join -** функция позволяет потоку дождаться потоку завершения другого потока. В более сложной ситуации, когда требуется дождаться завершения нескольких потоков, можно воспользоваться переменными условия. Описанная функция блокирует вызывающий поток до завершения указанного потокаю

3) **mutex.lock()** - метод пространства имён std::mutex,

предотвращающий доступ к разделяемому ресурсу нескольких потоков до тех пор, пока не будет вызван mutex.unlock().

4) **mutex.unlock()** - метод пространства имён std::mutex, который освобождает mutex и позволяет другим потокам получить доступ к разделяемому ресурсу.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pthread, pthread_create, pthread_join
2. Написать программу, которая будет удовлетворять заданию варианта.
3. Использовать pthread_create для распараллеливания выполнения программы.
4. При помощи функции threadFunc реализовать вычисление раунда метода Монте-Карло за поток.
5. Создать таймер при помощи библиотеки chrono и замерить время выполнения программы.
6. Скомпилировать обе программы при помощи cmake

Основные файлы программы

AssyncMain.cpp

```
#include <iostream>
#include <pthread.h>
#include <random>
#include <vector>
#include <mutex>
#include <chrono>

int rounds_count;
int count_same_rank;
int thread_count;
```

```
struct arg_t {
    int id;
    int thread_rounds_count;
};
```

```
//Один раунд метода
void Round() {
    std::random_device rd;
```

```
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(0, 51);
```

```
    std::vector<std::pair<int, int>> card_values(13);
    int index;
    for (int j = 0; j < 13; ++j) {
        card_values[j].first = j;
        card_values[j].second = 0;
    }
    int deck[52];
    for (int j = 0; j < 52; ++j) {
        deck[j] = j;
    }
```

```
    for (int j = 0; j < 52; ++j) {
        index = dis(gen);
        if (card_values[index % 13].second < 4) {
            card_values[index % 13].second += 1;
            std::swap(deck[j], deck[index]);
            continue;
        }
        index = dis(gen);
        card_values[index % 13].second += 1;
        std::swap(deck[j], deck[index]);
    }
```

```
    if (deck[0] % 13 == deck[1] % 13) {
        ++count_same_rank;
    }
}
```

```
//Подсчёт вероятности многопоточно
void* ThreadFunction(void* argument) {
    int* count = static_cast<int*>(argument);
    std::mutex mutex_count;
    for (int i = 0; i < *(count); ++i) {
        mutex_count.lock();
        Round();
        mutex_count.unlock();
    }
    pthread_exit(0);
}
```

```
int main(int argc, char* argv[]) {
    if (argc < 2) {
        throw std::logic_error("Указано неполное число ключей");
    }
```

```
    count_same_rank = 0;
    rounds_count = atoi(argv[1]);
    thread_count = atoi(argv[2]);
    if (rounds_count < thread_count) {
        thread_count = rounds_count;
    }
    if (thread_count <= 0) {
        throw std::logic_error("Количество потоков не может быть меньше 1");
    }
```

```
    arg_t args[thread_count];
    if (rounds_count % thread_count == 0) {
```

```
        for (int i = 0; i < thread_count; ++i) {
            args[i].id = i;
            args[i].thread_rounds_count = rounds_count / thread_count;
        }
    } else {
        for (int i = 0; i < thread_count - 1; ++i) {
            args[i].id = i;
            args[i].thread_rounds_count = rounds_count / thread_count;
        }
        args[thread_count - 1].id = thread_count - 1;
        args[thread_count - 1].thread_rounds_count = rounds_count / thread_count +
rounds_count % thread_count;
    }
```

```
    auto start = std::chrono::steady_clock::now();
```

```
    pthread_t tid[thread_count];
    for (int i = 0; i < thread_count; ++i) {
        pthread_create(&tid[i], nullptr, ThreadFunction, &args[i].thread_rounds_count);
    }
    for (int i = 0; i < thread_count; ++i) {
        pthread_join(tid[i], nullptr);
    }
    double probability = static_cast<double>(count_same_rank) / rounds_count;
    std::cout << probability << '\n';
```

```
    auto end = std::chrono::steady_clock::now();
    auto elapsed = std::chrono::duration_cast<std::chrono::milliseconds>(end - start);
    std::cout << elapsed.count() << '\n';
}
```

SyncMain.cpp

```
#include <iostream>
#include <pthread.h>
#include <random>
#include <vector>
#include <mutex>
#include <chrono>

//Функция вычисления вероятности однопоточно
double Probability(int rounds_count) {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(0, 51);
```

```
    int count_same_rank = 0;
```

```
    for (int i = 0; i < rounds_count; ++i) {
        std::vector<std::pair<int, int>> card_values(13);
        int index;
        for (int j = 0; j < 13; ++j) {
            card_values[j].first = j;
            card_values[j].second = 0;
        }
        int deck[52];
        for (int j = 0; j < 52; ++j) {
            deck[j] = j;
        }
```

```
        for (int j = 0; j < 52; ++j) {
            index = dis(gen);
            if (card_values[index % 13].second < 4) {
                card_values[index % 13].second += 1;
                std::swap(deck[j], deck[index]);
                continue;
            }
            index = dis(gen);
            card_values[index % 13].second += 1;
            std::swap(deck[j], deck[index]);
        }
```

```
        if (deck[0] % 13 == deck[1] % 13) {
            ++count_same_rank;
        }
    }
    double probability = static_cast<double>(count_same_rank) / rounds_count;
```

```
    return probability;
}
```

```
int main(int argc, char* argv[]) {
    if (argc == 1) {
        throw std::logic_error("Не указано число раундов");
    }
    auto start = std::chrono::steady_clock::now();
    double prob = Probability(atoi(argv[1]));
    auto end = std::chrono::steady_clock::now();
    auto elapsed = std::chrono::duration_cast<std::chrono::milliseconds>(end - start);
    std::cout << prob << '\n' << elapsed.count() << '\n';
}
```

Пример работы

dmitrijmrsh@LAPTOP-7SMT8REA:~/Labs-git/os_lab_2/src/build$ ./assync 1000000 12

0.058476

3793

dmitrijmrsh@LAPTOP-7SMT8REA:~/Labs-git/os_lab_2/src/build$ ./sync 1000000

0.058725

5566

Вывод

Во второй лабораторной работе я научился работать с потоками операционной системы. Изучив принципы работы потоков на низкоуровневом языке я реализовал вариант работы и сделал так, чтобы можно было выполнять вычисления, как на одном потоке, так и на нескольких. Используя системные вызовы я смог распараллелить программу и подсчитать, при помощи chrono, время выполнения программы. Как можно заметить на представленном ниже графике время выполнения на одном потоке большого количества вычисления производится медленнее, чем на 12 потоках, максимума моего процессора. Умение работать с потоками позволит в будущем более фундаментально понимать принципы работы много поточных программ.

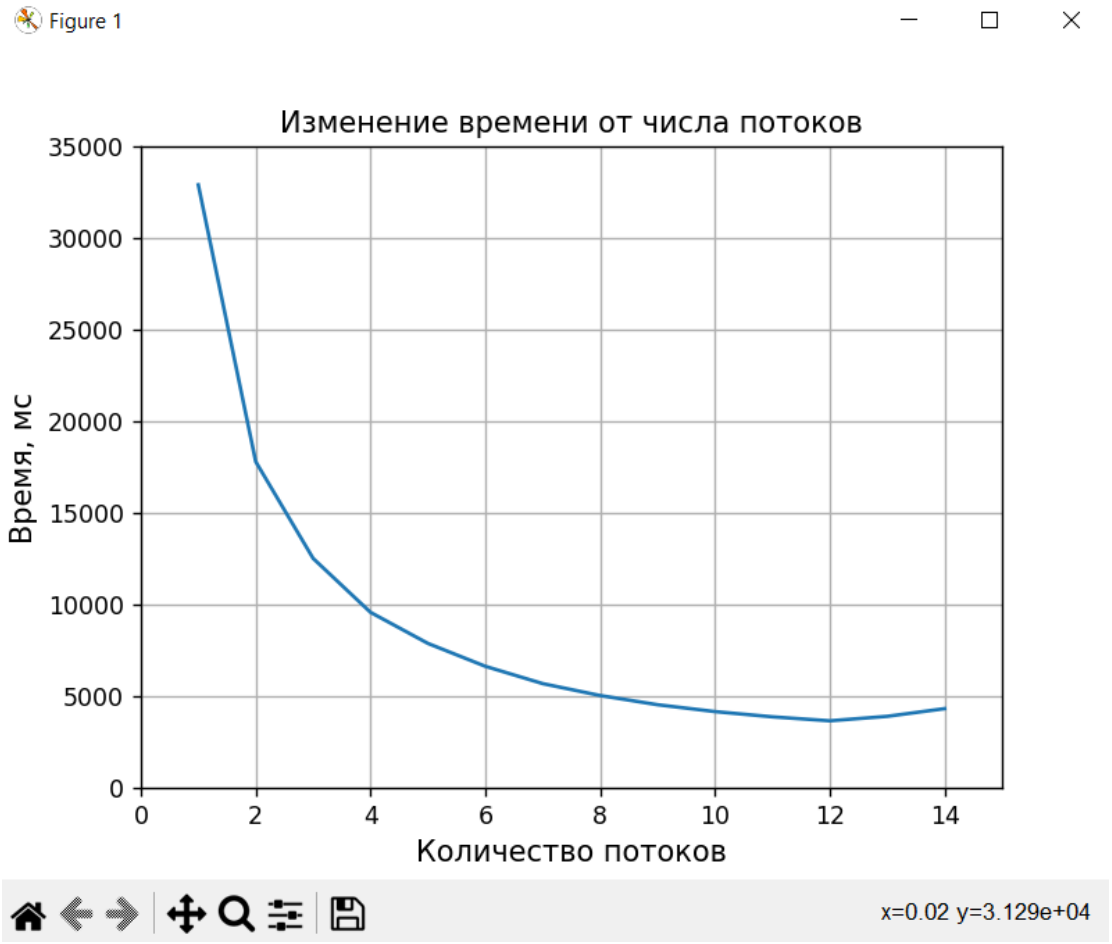
Figure 1
Изменение времени от числа потоков
Время, мс
Количество потоков
35000
30000
25000
20000
15000
10000
5000
0
0
2
4
6
8
10
12
14
x=0.02 y=3.129e+04