Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №1 по курсу
«Операционные системы»

УПРАВЛЕНИЕ ПОТОКАМИ

Студент: Железнов Илья Васильевич
Группа: М8О–210Б–22
Вариант: 7
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 7) Два человека играют в кости. Правила игры следующие: каждый игрок делает бросок 2-ух костей K раз; побеждает тот, кто выбросил суммарно большее количество очков. Задача программы экспериментально определить шансы на победу каждого из игроков. На вход программе подается K, какой сейчас тур, сколько очков суммарно у каждого из игроков и количество экспериментов, которые должна произвести программа

Общие сведения о программе

Программа компилируется при помощи утилиты g++ main.cpp -pthread и запускается путем запуска ./a.out <NUMBER OF THREAD>. Также используется заголовочные файлы: iostream, thread, chrono. В программе используются следующие системные вызовы:

1. **pthread_self** – Функция возвращает дескриптор потока, в которой эта функция была вызвана: pthread_t pthread_self(void); Ошибки для функции pthread_self() не определены.
2. **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.
3. **pthread_join** – Функция позволяет потоку дождаться завершения

другого потока. В более сложной ситуации, когда требуется дождаться завершения нескольких потоков, можно воспользоваться переменными условия. Функция pthread_join блокирует вызывающий поток до завершения указанного потока.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pthread, pthread_self, 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 бросков кубиков игроками за поток.
5. Создать таймер при помощи библиотеки chrono и замерить время выполнения программы.
6. Скомпилировать обе программы при помощи g++ main.cpp -pthread и запустить ./a.out <NUMBER OF THREADS>.

Основные файлы программы

main.cpp:

```
#include <iostream>
#include <thread>
#include <chrono>

typedef struct _thread_data {
    int numberOfPointsFirst;
    int numberOfPointsSecond;
    int plays;
    int tour;
    int experements;
    int winChanceFirst;
    int winChanceSecond;
} threadData;

void* threadFunc(void* arg)
{
    threadData* tData = (threadData*)arg;
    unsigned int seed;
    seed = pthread_self();

    int firstP, secondP;
    for (int i = 0; i < tData->experements; ++i) {
        firstP = tData->numberOfPointsFirst;
        secondP = tData->numberOfPointsSecond;
        for (int j = 0; j < tData->plays - tData->tour + 1; ++j) { // + 1, тк индексация с 0
            firstP += rand_r(&seed) % 6 + 1;
            firstP += rand_r(&seed) % 6 + 1;
            secondP += rand_r(&seed) % 6 + 1;
            secondP += rand_r(&seed) % 6 + 1;
        }

        if (firstP > secondP) {
```

```
            tData->winChanceFirst++;
        }

        if (secondP > firstP) {
            tData->winChanceSecond++;
        }
    }

    return 0;
}

int main(int argc, char* argv[])
{
    if (argc != 2) {
        std::cerr << "\tError! Program must have only 1 key\n EXAMPLE:\n \t"
                << argv[0] << " <NUMBER_OF_THREADS>\n";
        exit(EXIT_FAILURE);
    }

    clock_t startTime, endTime;
    float timer;
    startTime = clock();

    int numberOfThreads = atoi(argv[1]);

    int numberOfPointsFirst, numberOfPointsSecond,
    plays, tour, experements;
    float percentWinsFirst = 0, percentWinsSecond = 0;

    std::cout << "1) Enter number of points first player: ";
    std::cin >> numberOfPointsFirst;
    std::cout << "2) Enter number of points second player: ";
    std::cin >> numberOfPointsSecond;
    std::cout << "3) Enter the number of this tour: ";
    std::cin >> tour;
    std::cout << "4) Enter number of throws (K): ";
    std::cin >> plays;
    std::cout << "5) Enter number of experements: ";
    std::cin >> experements;

    int numbOfExperemForOneThread = experements / numberOfThreads;

    threadData tData[numberOfThreads];

    for (int i = 0; i < numberOfThreads; ++i) {
        tData[i].numberOfPointsFirst = numberOfPointsFirst;
        tData[i].numberOfPointsSecond = numberOfPointsSecond;
        tData[i].plays = plays;
        tData[i].tour = tour;

        if (i == numberOfThreads - 1) {
            tData[i].experements = numbOfExperemForOneThread + experements % numberOfThreads;
        } else {
            tData[i].experements = numbOfExperemForOneThread;
        }

        tData[i].winChanceFirst = 0;
        tData[i].winChanceSecond = 0;
    }
```

```
    pthread_t th[numberOfThreads];

    for (int i = 0; i < numberOfThreads; ++i) {
        if (pthread_create(&th[i], NULL, threadFunc, &tData[i]) != 0) {
            std::cerr << "\tError! Can'not create thread # " << i << std::endl;
            exit(EXIT_FAILURE);
        }
    }

    for (int i = 0; i < numberOfThreads; ++i) {
        if (pthread_join(th[i], NULL) != 0) {
            std::cerr << "Error! Can'not join thread # " << i << std::endl;
            break;
        }

        threadData* res = &tData[i];
        percentWinsFirst += res->winChanceFirst;
        percentWinsSecond += res->winChanceSecond;
    }

    std::cout << "Probability of the first player to win: "
              << percentWinsFirst / experements << std::endl;
    std::cout << "Probability of the second player to win: "
              << percentWinsSecond / experements << std::endl;

    endTime = clock();
    timer = endTime - startTime;
    std::cout << "Time: " << timer / CLOCKS_PER_SEC << std::endl;

    return 0;
}
```

Пример работы

```
keinpop@DESKTOP-T6SLHUS:/mnt/c/oc_lab/lab2/src$
./a.out 1
1) Enter number of points first player: 0
2) Enter number of points second player: 0
3) Enter the number of this tour: 1
4) Enter number of throws (K): 1000
5) Enter number of experements: 10000
Probability of the first player to win: 0.5019
Probability of the second player to win: 0.4946
Time: 0.126852
keinpop@DESKTOP-T6SLHUS:/mnt/c/oc_lab/lab2/src$
./a.out 100
1) Enter number of points first player: 0
```

```
2) Enter number of points second player: 0
3) Enter the number of this tour: 1
4) Enter number of throws (K): 1000
5) Enter number of experements: 10000
Probability of the first player to win: 0.4955
Probability of the second player to win: 0.5011
Time: 0.044847
```

Вывод

В 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 время выполнения на одном потоке большого количества вычисления производится медленнее, чем на 100 потоках. Умение работать с потоками позволит в будущем более фундаментально понимать принципы работы много поточных программ.