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: Меркулов Ф. А.

Группа: М8О-207Б-21

Вариант: 7

Преподаватель: Черемисинов Максим Леонидович

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/WhatTheMUCK/OSi

**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 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.

7 вариант) 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 из файла main.cpp. Также используется заголовочные файлы: stdio.h, stdlib.h, iostream, pthread.h

В программе используются следующие системные вызовы:

1. pthread\_create() – создаёт новый поток
2. pthread\_join(pthread\_t THREAD\_ID, void \*\* DATA) - ожидает завершения потока обозначенного THREAD\_ID

**Общий метод и алгоритм решения**

С помощью ключа запуска передаётся необходимое количество потоков. С консоли программа считывает: количество очков первого игрока, количество очков второго игрока, тур, количество бросков костей и общее количество экспериментов. На каждый поток передаётся количество экспериментов равное [общее количество экспериментов / количество потоков], но так как количество экспериментов на потоках – целое число, то последний поток обрабатывает [(общее количество экспериментов / количество потоков) + (общее количество экспериментов % количество потоков)]. На каждом потоке происходит экспериментальный подсчёт количества побед первого и второго игроков, с помощью потокобезопасной функции rand\_r(&seed), где seed – id потока. То есть для каждого отдельного эксперимента, для каждого игрока программа моделирую бросок 2 кубиков заданное количество раз и выясняет кто победил в этом эксперименте: первый или второй игрок. Так все потоки подсчитывают общее количество побед первого и второго игроков за общее количество экспериментов и для нахождения вероятности победы, мы просто делим число побед игрока на общее число экспериментов.

**Исходный код**

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <pthread.h>

//#include <time.h>

using namespace std;

typedef struct thread\_data{

int counter\_first;

int counter\_second;

int plays;

int tour;

int experements;

int wins\_first;

int wins\_second;

} thread\_data;

void\* thread\_func(void \*arg){

thread\_data \*tdata = (thread\_data \*)arg;

unsigned int seed;

seed = pthread\_self();

int first;

int second;

for (int counter1 = 0; counter1 < tdata->experements; counter1++){

first = tdata->counter\_first;

second = tdata->counter\_second;

for (int counter2 = 0; counter2 < tdata->plays - tdata->tour + 1; counter2++){

first += rand\_r(&seed)%6 + 1;

first += rand\_r(&seed)%6 + 1;

second += rand\_r(&seed)%6 + 1;

second += rand\_r(&seed)%6 + 1;

}

if (first > second)

tdata->wins\_first++;

if (second > first)

tdata->wins\_second++;

}

return 0;

}

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

if (argc != 2){

cerr << "error: Program must have only 1 key\n";

} else {

//clock\_t start\_time, end\_time;

//float timer;

//start\_time = clock();

int NUMBER\_THREADS = atoi(argv[1]);

//cout << "#Количество потоков = " << NUMBER\_THREADS << "\n";

int counter\_first, counter\_second, plays, tour, experements, count\_of\_experements\_for\_one\_thread;

float wins\_first = 0, wins\_second = 0;

cout << "Введите:\n 1)Количество очков первого игрока: ";

cin >> counter\_first;

cout << " 2)Количество очков второго игрока: ";

cin >> counter\_second;

cout << " 3)Номер данного тура: ";

cin >> tour;

cout << " 4)Количество бросков костей: ";

cin >> plays;

cout << " 5)Количество экспериментов: ";

cin >> experements;

count\_of\_experements\_for\_one\_thread = experements / NUMBER\_THREADS;

//cout << "#Количество эксперементов для одного потока = " << count\_of\_experements\_for\_one\_thread << "\n";

thread\_data tdata[NUMBER\_THREADS];

for (int i = 0; i < NUMBER\_THREADS; i++){

tdata[i].counter\_first = counter\_first;

tdata[i].counter\_second = counter\_second;

tdata[i].plays = plays;

tdata[i].tour = tour;

if (i == NUMBER\_THREADS - 1)

tdata[i].experements = count\_of\_experements\_for\_one\_thread + experements % NUMBER\_THREADS;

else

tdata[i].experements = count\_of\_experements\_for\_one\_thread;

tdata[i].wins\_first = 0;

tdata[i].wins\_second = 0;

}

pthread\_t thread[NUMBER\_THREADS];

for (int i = 0; i < NUMBER\_THREADS; i++){

if(pthread\_create(&thread[i], NULL, thread\_func, &tdata[i]) != 0){

cerr << "error: Cannot create thread # " << i << "\n";

break;

}

}

for (int i = 0; i < NUMBER\_THREADS; i++){

if(pthread\_join(thread[i], NULL) != 0){

cerr << "error: Cannot join thread # " << i << "\n";

break;

}

thread\_data \*result = &tdata[i];

wins\_first += result->wins\_first;

wins\_second += result->wins\_second;

}

cout << "Вероятность победы первого игрока: " << wins\_first/experements << "\n";

cout << "Вероятность победы второго игрока: " << wins\_second/experements << "\n";

//end\_time = clock()`;

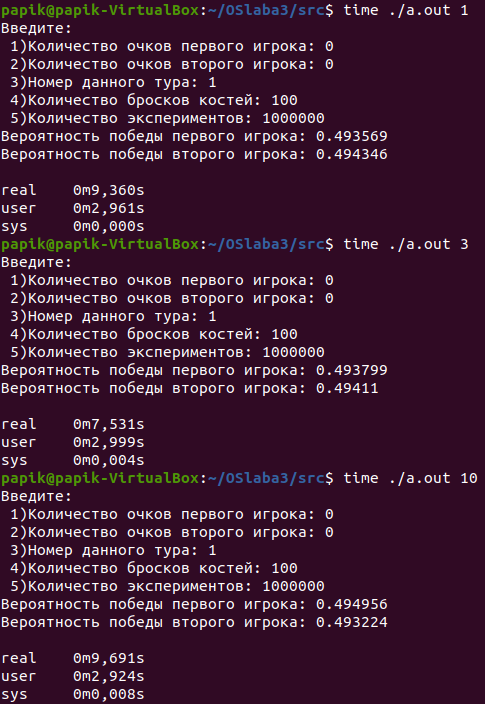
//timer = end\_time - start\_time;

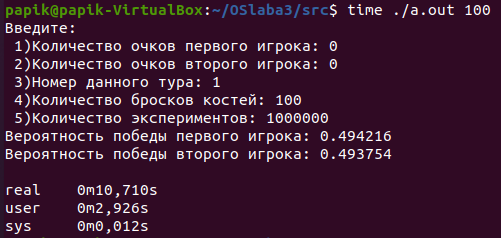
//cout << "Time: " << timer / CLOCKS\_PER\_SEC << "\n";

}

}

**Демонстрация работы программы**



****

Можем заметить, что выполнение одного и того же объёма данных происходит быстрее при увеличении числа потоков примерно до количества ядер процессора (в моём устройстве их 4) и медленее при числе потоков, превышающем количество ядер. Это происходит в связи с тем, что нужно некоторое время для создания потоков и выигрыша во времени не происходит, если нельзя выполнять потоки обособленно на разных ядрах.

**Выводы**

За время выполнения лабораторной работы я научился управлять потоками в ОС, а также разобрался с обеспечение синхронизации между потоками.

.