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

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";
    }
}
```

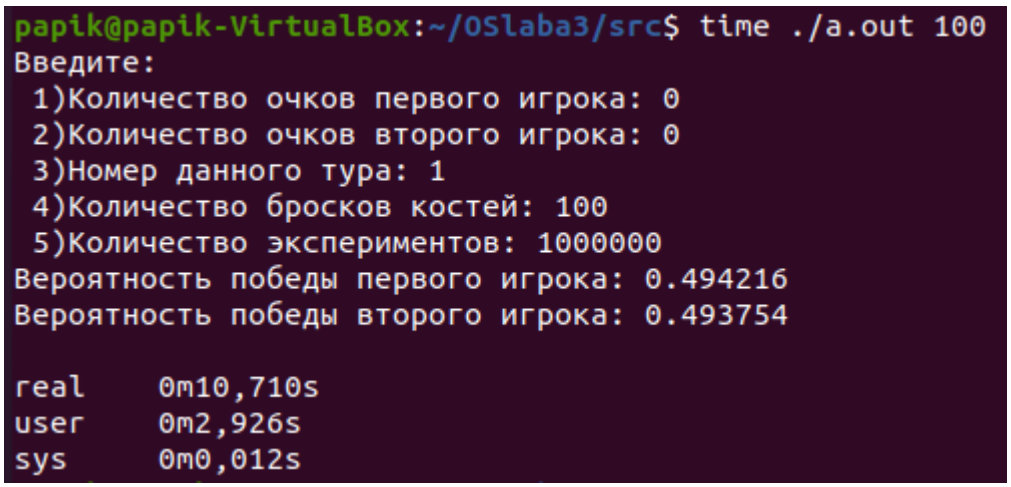
Демонстрация работы программы

```
papik@papik-VirtualBox:~/OSlaba3/src$ time ./a.out 1
Введите:
 1)Количество очков первого игрока: 0
 2)Количество очков второго игрока: 0
 3)Номер данного тура: 1
 4)Количество бросков костей: 100
 5)Количество экспериментов: 1000000
Вероятность победы первого игрока: 0.493569
Вероятность победы второго игрока: 0.494346

real    0m9,360s
user    0m2,961s
sys     0m0,000s
papik@papik-VirtualBox:~/OSlaba3/src$ time ./a.out 3
Введите:
 1)Количество очков первого игрока: 0
 2)Количество очков второго игрока: 0
 3)Номер данного тура: 1
 4)Количество бросков костей: 100
 5)Количество экспериментов: 1000000
Вероятность победы первого игрока: 0.493799
Вероятность победы второго игрока: 0.49411

real    0m7,531s
user    0m2,999s
sys     0m0,004s
papik@papik-VirtualBox:~/OSlaba3/src$ time ./a.out 10
Введите:
 1)Количество очков первого игрока: 0
 2)Количество очков второго игрока: 0
 3)Номер данного тура: 1
 4)Количество бросков костей: 100
 5)Количество экспериментов: 1000000
Вероятность победы первого игрока: 0.494956
Вероятность победы второго игрока: 0.493224

real    0m9,691s
user    0m2,924s
sys     0m0,008s
```



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.

.