Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**РАБОТА С ОБЩЕЙ ПАМЯТЬЮ**

Студент: Баталин Дмитрий Андреевич

Группа: М8О–212Б–20

Вариант: 7

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством каналов

## 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.

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.c с помощью утилиты CMake. Внутри программы 2 функции: int main(int argc, char \*\*argv), принимающая аргументы командной строки, и функция void\* experiment(void \*param), которая используется для проведения эксперимента по вычислению количества побед.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы мютексов и потоков в си.
2. Реализовать функцию, проводящую эксперимент: функция должна принимать все заданные параметры игры и имитировать бросок четырех костей (каждый игрок по 2 кости) k раз, где k = experiment\_number / threads\_number, где experiment\_number - количество заданных игроком экспериментов, a threads\_number -количество потоков. Таким образом, мы сможем запустить функцию threads\_number раз, в каждой из которых experiment\_number раз проведем эксперимент.
3. Реализовать простейшее вычисление вероятностей победы каждого игрока.
4. Проверить, какой вариант решения задачи в данном случае лучше: однопоточное решение или многопоточное.

**Основные файлы программы**

**main.c:**

#include "stdio.h"

#include "stdlib.h" // pid\_t declaration is here

#include "unistd.h" // fork() and getpid() declarations are here

#include "pthread.h"

#include <sys/time.h> // for clock\_t, clock(), CLOCKS\_PER\_SEC

#include <string.h>

**pthread\_mutex\_t** player1\_mutex;

**pthread\_mutex\_t** player2\_mutex;

**typedef** **struct**

{

**int** experiment\_num;

**int** throws\_number;

**int** tour\_number;

**int** player1\_start\_points;

**int** player2\_start\_points;

**int** player1\_wins;

**int** player2\_wins;

} game\_parameters;

**void**\* **experiment**(**void** \*param) {

game\_parameters \*game = (game\_parameters\*) param;

**int** player1\_wins = **0**;

**int** player2\_wins = **0**;

// printf("experiment\_num: %d", game->experiment\_num);

**for** (**int** j = **0**; j < game->experiment\_num; j++) {

**int** player1\_experiment\_points = game->player1\_start\_points;

**int** player2\_experiment\_points = game->player2\_start\_points;

**for** (**int** i = game->tour\_number; i < game->throws\_number; i++) {

player1\_experiment\_points += rand() % **12** + **1**;

player2\_experiment\_points += rand() % **12** + **1**;

}

**if** (player1\_experiment\_points != player2\_experiment\_points) {

**if** (player1\_experiment\_points > player2\_experiment\_points) {

player1\_wins += **1**;

}

**else** {

player2\_wins += **1**;

}

}

}

// printf("PLayer1 wins: %d\n", player1\_wins);

// printf("PLayer2 wins: %d\n\n", player2\_wins);

pthread\_mutex\_lock(&player1\_mutex);

game->player1\_wins += player1\_wins;

pthread\_mutex\_unlock(&player1\_mutex);

pthread\_mutex\_lock(&player2\_mutex);

game->player2\_wins += player2\_wins;

pthread\_mutex\_unlock(&player2\_mutex);

pthread\_exit(**0**);

}

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

srand(time(NULL));

**int** threads\_number = **1**;

**if** ((argc > **3**) || (argc == **2**)) {

perror("Wrong number of parameters");

printf("You wrote %d parametres, but it must be only 2 or 0", argc);

**return** **2**;

}

// for (int i = 0; i < argc; i++) {

// printf("%d param: %s\n", i, argv[i]);

// }

**if** (argc == **3**) {

**if** (strcmp(argv[**1**], "-t") == **0**) {

threads\_number = atoi(argv[**2**]);

}

**else** {

perror("No such parameter exists");

**return** **3**;

}

}

**int** experiment\_number = **1**;

game\_parameters game = {**0**};

pthread\_mutex\_init(&player1\_mutex, NULL);

pthread\_mutex\_init(&player2\_mutex, NULL);

printf("Enter the number of throws (K-number): ");

scanf("%d", &game.throws\_number);

printf("Enter the tour number: ");

scanf("%d", &game.tour\_number);

printf("Enter the amount of points of the first player: ");

scanf("%d", &game.player1\_start\_points);

printf("Enter the amount of points of the second player: ");

scanf("%d", &game.player2\_start\_points);

printf("Enter the number of experiments: ");

scanf("%d", &experiment\_number);

**if** (experiment\_number < threads\_number) {

perror("The number of threads is more than number of experiments");

**return** **4**;

}

game.experiment\_num = experiment\_number / threads\_number;

**pthread\_t** tid[threads\_number];

**struct** timeval start\_time, end\_time;

**long** seconds, microseconds;

gettimeofday(&start\_time, NULL);

**for** (**int** i=**0**; i<threads\_number; i++)

{

pthread\_create(&tid[i], NULL, experiment, &game);

}

**for** (**int** i=**0**; i<threads\_number; i++)

{

pthread\_join(tid[i], NULL);

}

gettimeofday(&end\_time, NULL);

seconds = end\_time.tv\_sec - start\_time.tv\_sec;

microseconds = end\_time.tv\_usec - start\_time.tv\_usec;

**double** elapsed\_time = seconds + microseconds / **1e6**;

**double** player1\_chance = (**double**)game.player1\_wins / (**double**)(game.player1\_wins + game.player2\_wins);

**double** player2\_chance = (**double**)game.player2\_wins / (**double**)(game.player1\_wins + game.player2\_wins);

printf("First player chance to win: %f**\n**", player1\_chance);

printf("Second player chance to win: %f**\n**", player2\_chance);

printf("The elapsed time is %f seconds**\n**", elapsed\_time);

**return** **0**;

}

**Пример работы**

moses@moses:~/cs/os/lab2/src/build$ cat ../../test/test5.txt

200

1

2

2

1000moses@moses:~/cs/os/lab2/src/build$ ./lab2 -t 1 < ../../test/test5.txt

Enter the number of throws (K-number):

Enter the tour number:

Enter the amount of points of the first player:

Enter the amount of points of the second player:

Enter the number of experiments:

First player chance to win: 0.506533

Second player chance to win: 0.493467

The elapsed time is 0.011005 seconds

moses@moses:~/cs/os/lab2/src/build$ ./lab2 -t 10 < ../../test/test5.txt

Enter the number of throws (K-number):

Enter the tour number:

Enter the amount of points of the first player:

Enter the amount of points of the second player:

Enter the number of experiments:

First player chance to win: 0.522704

Second player chance to win: 0.477296

The elapsed time is 0.104480 seconds

moses@moses:~/cs/os/lab2/src/build$ ./lab2 -t 20 < ../../test/test5.txt

Enter the number of throws (K-number):

Enter the tour number:

Enter the amount of points of the first player:

Enter the amount of points of the second player:

Enter the number of experiments:

First player chance to win: 0.514573

Second player chance to win: 0.485427

The elapsed time is 0.102451 seconds

moses@moses:~/cs/os/lab2/src/build$ ./lab2 -t 30 < ../../test/test5.txt

Enter the number of throws (K-number):

Enter the tour number:

Enter the amount of points of the first player:

Enter the amount of points of the second player:

Enter the number of experiments:

First player chance to win: 0.491371

Second player chance to win: 0.508629

The elapsed time is 0.056475 seconds

moses@moses:~/cs/os/lab2/src/build$ cat ../../test/test1.txt

20

1

0

0

1moses@moses:~/cs/os/lab2/src/build$ ./lab2 -t 1 < ../../test/test1.txt

Enter the number of throws (K-number):

Enter the tour number:

Enter the amount of points of the first player:

Enter the amount of points of the second player:

Enter the number of experiments:

First player chance to win: 0.000000

Second player chance to win: 1.000000

The elapsed time is 0.000423 seconds

moses@moses:~/cs/os/lab2/src/build$ ./lab2 -t 1 < ../../test/test1.txt

Enter the number of throws (K-number):

Enter the tour number:

Enter the amount of points of the first player:

Enter the amount of points of the second player:

Enter the number of experiments:

First player chance to win: 0.000000

Second player chance to win: 1.000000

The elapsed time is 0.000379 seconds

moses@moses:~/cs/os/lab2/src/build$ ./lab2 -t 1 < ../../test/test1.txt

Enter the number of throws (K-number):

Enter the tour number:

Enter the amount of points of the first player:

Enter the amount of points of the second player:

Enter the number of experiments:

First player chance to win: 1.000000

Second player chance to win: 0.000000

The elapsed time is 0.000365 seconds

**Вывод**

Анализируя результаты выполненной работы, можно сказать, что увеличение количества потоков не даёт мне выигрыша во времени, а только ухудшает результат и производительность программы. Взглянем на результаты с пятым тестом:

|  |  |
| --- | --- |
| **Количество потоков** | **Время выполнения, секунд** |
| 1 | 0.011005 |
| 2 | 0.271034 |
| 3 | 0.035536 |
| 10 | 0.104480 |
| 20 | 0.102451 |
| 30 | 0.056475 |
| 40 | 0.060141 |
| 50 | 0.071365 |
| 100 | 0.074344 |

Из данной таблицы следует, что программа, работающая с одним потоком, выполняется эффективнее всего. Также из таблицы можно выяснить, что программа работает менее эффективно с 10 и 20 потоками, чем с 30. При увеличении количества потоков, производительность падает. Предположу, что связано это с тратой ресурсов компьютера на создание и выход из потоков, что занимает больше времени, чем просто выполнение программы однопоточно и не асинхронно.