**Московский авиационный институт (национальный исследовательский университет)**

Институт №8 «Информационные технологии и прикладная математика» Кафедра 806 «Вычислительная математика и программирование» Дисциплина «Операционные системы»

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Управление потоками в ОС**

## Студент: Кондратьев Егор Алексеевич

## Группа: М80-206Б-19

Преподаватель: Миронов Е.С.

Дата:

## Оценка:

Москва, 2021

## **Постановка задачи**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы(Windows/Linux). Ограничение потоков должно быть задано ключом запуска вашей программ.

Привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Получившиеся результаты объяснить.

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

## **Метод решения**

*Используемые системные и библиотечные вызовы для выполнения работы:*

pid\_t **pthread\_create**(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*thread\_function) (void), void \*arg) данная функция запускает новый поток в вызывающем процессе. Новый поток начинает исполнение, вызывая thread\_function (void \*arg).

pid\_t **pthread\_join**(pthread\_t thread, void \*\*retval) - данная функция ожидает завершения указанного потока. Если этот поток уже завершен, то данная функция немедленно возвращается.

### *Краткий алгоритм решения поставленной задачи:*

1. Происходит считывание аргументов командной строки
2. Создание массива параметров для каждого потока
3. Происходит запуск необходимого числа потоков:
   1. Если количество экспериментов указано меньше, чем потоков, то количество потоков уменьшается до количества экспериментов
   2. Если количество экспериментов указано больше, чем потоков, то количество потоков не меняется
4. Создание заданного числа потоков
   1. Моделировать броски кубика
   2. Подсчет выигранных экспериментов того или иного игрока
   3. Запись результата в переданный в thread\_function аргумент params
5. Ожидание завершения потоков
6. Подсчет итоговой вероятности
7. Освобождение памяти под массивы потоков и параметров

### *Метрики параллельных вычислений*

Метрики параллельных вычислений используются для оценки роста производительности, получаемого при параллельном решении задачи на p процессах. Также они позволяют определить необходимое количество процессоров, используемых для решения конкретной задачи. Метрики будут приводится для двух случаев: для случая использования встроенного генератора псевдослучайных чисел, и для случая использования своего генератора псевдослучайных чисел.

* Tp – время выполнения на p различных потоках/вычислительных ядрах
* Sp – ускорение. Sp = T1/Tp, Sp < p
* Xp – эффективность/загруженность. Xp = Sp/p, Xp<1

## **Тестирование**

| **p** | **T1** (sec) | **Tp** (sec) | Ускорение  (**Sp**=T1/Tp) | Эффективность  (**Xp**=Sp/p) |
| --- | --- | --- | --- | --- |
| 2 | 0.040996 | 0.045936 | 0.892473 | 0.446237 |
| 4 | 0.040996 | 0.036755 | 1.115406 | 0.371802 |
| 8 | 0.040996 | 0.031698 | 1.293361 | 0.323340 |
| 16 | 0.040996 | 0.024488 | 1.674157 | 0.334831 |
| 32 | 0.040996 | 0.025303 | 1.620192 | 0.270032 |
| 64 | 0.040996 | 0.024713 | 1.658926 | 0.236989 |
| 128 | 0.040996 | 0.025178 | 1.628231 | 0.203529 |
| 256 | 0.040996 | 0.030967 | 1.323881 | 0.147098 |
| 512 | 0.040996 | 0.026217 | 1.563745 | 0.156375 |
| 1024 | 0.040996 | 0.028532 | 1.436876 | 0.130625 |

## **Вывод консоли**

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_3/src$ gcc -pthread -o main main.c

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_3/src$ ./main

usage: threads\_number K tour\_now first\_player\_score second\_player\_score experiments

root@Du:/mnt/c/Users/egork/Desktop/os/lab\_3/src$ ./main 5000 100 1 0 0 10000

First chances: 5049/10000 = 50.490000%

Second chances: 4826/10000 = 48.260000%

## **Листинг программы**

#include <pthread.h>

#include <string.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define min(x, y) (x) < (y)? (x) : (y)

#define COUNT\_ARGS 7

typedef u\_int64\_t count\_t;

typedef struct \_player{

count\_t scores;

} Player;

typedef struct \_params {

Player first, second;

count\_t experiments;

count\_t moves\_count;

count\_t first\_won, second\_won;

} Params;

typedef struct \_command {

int threads\_num;

int K;

int tour\_now;

int first\_player\_score, second\_player\_score;

int experiments;

} Command;

enum WON{

DRAW,

FIRST,

SECOND

};

int rand\_score(){

int (\*rfunc)(void) = rand;

return (rfunc() % 6) + (rfunc() % 6) + 2;

}

int experiment(count\_t moves\_count, Player first, Player second){

for (int i = 0; i < moves\_count; ++i) {

first.scores += rand\_score();

second.scores += rand\_score();

}

if(first.scores > second.scores){

return FIRST;

}

else if(second.scores > first.scores) {

return SECOND;

}

return DRAW;

}

void init(Params \*ptr, Command \*command)

{

Player f = {command->first\_player\_score};

Player s = {command->second\_player\_score};

ptr[0].first = f;

ptr[0].second = s;

ptr[0].first\_won = ptr[0].second\_won = 0;

ptr[0].experiments = command->experiments / command->threads\_num;

ptr[0].moves\_count = (command->K - command->tour\_now - 1);

for (size\_t i = 1; i < command->threads\_num; ++i) {

ptr[i].first = ptr[i-1].first;

ptr[i].second = ptr[i-1].second;

ptr[i].moves\_count = ptr[i - 1].moves\_count;

ptr[i].experiments = ptr[i - 1].experiments;

ptr[i].first\_won = ptr[i].second\_won = 0;

}

ptr[command->threads\_num - 1].experiments += command->experiments % command->threads\_num;

}

void \*thread\_function(void \*somePtr){

srand(time(NULL));

Params \*params = (Params \*)somePtr;

for (int i = 0; i < params->experiments; ++i) {

int won = experiment(params->moves\_count, params->first, params->second);

if(won == FIRST){

params->first\_won++;

} else if(won == SECOND){

params->second\_won++;

}

}

return NULL;

}

void parse\_command\_line(int argc, char \*\*argv, Command \*command) {

if(argc != COUNT\_ARGS){

fprintf(stderr, "%s\n", "usage: threads\_number K tour\_now first\_player\_score second\_player\_score experiments");

exit(EXIT\_FAILURE);

}

command->threads\_num = atoi(argv[1]);

command->K = atoi(argv[2]);

command->tour\_now = atoi(argv[3]);

command->first\_player\_score = atoi(argv[4]);

command->second\_player\_score = atoi(argv[5]);

command->experiments = atoi(argv[6]);

}

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

Command command;

parse\_command\_line(argc, argv, &command);

command.threads\_num = min(command.threads\_num, command.experiments);

pthread\_t \*pthread\_ids = (pthread\_t \*)calloc(command.threads\_num, sizeof(pthread\_t));

Params \*params = (Params \*)calloc(command.threads\_num, sizeof(Params));

init(params, &command);

for (int i = 0; i < command.threads\_num; ++i) {

pthread\_create(&pthread\_ids[i], NULL, thread\_function, (void \*) &params[i]);

}

for (int j = 0; j < command.threads\_num; ++j) {

pthread\_join(pthread\_ids[j], NULL);

}

int first\_won = 0;

int second\_won = 0;

for (int i = 0; i < command.threads\_num; ++i) {

first\_won += params[i].first\_won;

second\_won += params[i].second\_won;

}

printf("First chances: %d/%d = %LF%%\n", first\_won, command.experiments, (long double)first\_won / command.experiments \* 100);

printf("Second chances: %d/%d = %LF%%\n", second\_won, command.experiments, (long double)second\_won / command.experiments \* 100);

free(pthread\_ids);

free(params);

return 0;

}

## **Выводы**

Выполнив данную лабораторную работу, научился методам работы с многопоточностью; узнал, что создать поток быстрее, чем создать процесс; познакомился с метриками, по которым можно оценивать эффективность и ускорение программы относительно однопоточной.

Потоки помогают, распараллелить несколько разных вычислений, однако большое количество потоков может плохо отразиться на скорости вычислений в этих самых потоках.