Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Группа: М8О-211Б-23

Студент: Грищенко В.А.

Преподаватель: Бахарев В.Д.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 29.10.24

Москва, 2024

**Постановка задачи**

**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

**Общий метод и алгоритм решения**

Использованные системные вызовы:

1. **pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start\_routine) (void \*), void \*arg);** – создает новый поток**.**
2. **pthread\_join(pthread\_t thread, void \*\*retval**); – ожидает завершения указанного потока.
3. **pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \*attr); –** инициализирует мьютекс.
4. **pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex**); – разрушает мьютекс.
5. **pthread\_mutex\_lock(pthread\_mutex\_t \*mutex**); – блокирует мьютекс
6. **pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex**); – разблокирует мьютекс
7. **clock\_gettime(clockid\_t clk\_id, struct timespec \*tp);** – получает текущее время

Описание лабораторной работы

В рамках лабораторной работы была разработана многопоточная программа на языке C, которая моделирует игру в кости и определяет шансы на победу каждого из игроков на основе экспериментальных данных. Программа использует POSIX Threads (pthreads) для параллельного выполнения симуляций игр и потокобезопасный массив для хранения результатов. Программа также включает таймер для измерения времени выполнения.

**Цель лабораторной работы**

Целью лабораторной работы было изучение и применение многопоточности с использованием POSIX Threads (pthreads) для параллельного выполнения задач, а также использование потокобезопасных структур данных для хранения и обработки результатов.

**Описание программы**

Программа моделирует игру в кости, в которой два игрока делают броски двух костей K раз. Побеждает тот, кто набрал большее количество очков. Программа принимает параметры через аргументы командной строки, включая количество бросков K, текущий тур, суммарное количество очков у каждого игрока, количество экспериментов и количество потоков.

Основные компоненты программы:

1)Функция roll\_dice: Генерирует случайное значение для броска двух костей.

2)Функция simulate\_game: Симулирует одну игру, возвращая результат (победа первого игрока, победа второго игрока или ничья).

3)Функция simulate\_games: Потоковая функция, которая выполняет симуляции игр и записывает результаты в массив.

4)Основная функция main:

Принимает параметры через аргументы командной строки.

Валидирует входные данные.

Создает потоки для параллельного выполнения симуляций.

Объединяет результаты из всех потоков.

Выводит шансы на победу каждого игрока и время выполнения программы.

**Код программы**

Main.c

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <time.h>

#include <unistd.h>

typedef struct {

int K;

int currentTurn;

int player1\_total;

int player2\_total;

int experiments;

int \*results;

} GameParams;

int roll\_dice(unsigned int \*seed) {

return (rand\_r(seed) % 6 + 1) + (rand\_r(seed) % 6 + 1);

}

int simulate\_game(int K, unsigned int \*seed) {

int player1\_score = 0;

int player2\_score = 0;

for (int i = 0; i < K; i++) {

player1\_score += roll\_dice(seed);

player2\_score += roll\_dice(seed);

}

return player1\_score > player2\_score ? 1 : (player1\_score < player2\_score ? 2 : 0);

}

void \*simulate\_games(void \*arg) {

GameParams \*params = (GameParams \*)arg;

unsigned int seed = (unsigned int)time(NULL) ^ pthread\_self();

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

params->results[i] = simulate\_game(params->K, &seed);

}

return NULL;

}

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

if (argc != 7) {

printf("Usage: %s <K> <current turn> <player1 total score> <player2 total score> <number of experiments> <number of threads>\n", argv[0]);

return 1;

}

int K = atoi(argv[1]);

int currentTurn = atoi(argv[2]);

int player1\_total = atoi(argv[3]);

int player2\_total = atoi(argv[4]);

int experiments = atoi(argv[5]);

int num\_threads = atoi(argv[6]);

if (K <= 0 || currentTurn < 0 || player1\_total < 0 || player2\_total < 0 || experiments <= 0 || num\_threads <= 0) {

printf("Invalid input data. All parameters must be positive integers.\n");

return 1;

}

int experiments\_per\_thread = experiments / num\_threads;

int remaining\_experiments = experiments % num\_threads;

pthread\_t threads[num\_threads];

GameParams params[num\_threads];

int \*results = (int \*)malloc(experiments \* sizeof(int));

struct timespec start, end;

clock\_gettime(CLOCK\_MONOTONIC, &start);

for (int i = 0; i < num\_threads; i++) {

params[i].K = K;

params[i].currentTurn = currentTurn;

params[i].player1\_total = player1\_total;

params[i].player2\_total = player2\_total;

params[i].experiments = experiments\_per\_thread + (i < remaining\_experiments ? 1 : 0);

params[i].results = results + (i \* experiments\_per\_thread + (i < remaining\_experiments ? i : remaining\_experiments));

pthread\_create(&threads[i], NULL, simulate\_games, &params[i]);

}

for (int i = 0; i < num\_threads; i++) {

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

}

clock\_gettime(CLOCK\_MONOTONIC, &end);

double time\_taken = (end.tv\_sec - start.tv\_sec) + (end.tv\_nsec - start.tv\_nsec) / 1e9;

int player1\_wins = 0;

int player2\_wins = 0;

int draws = 0;

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

if (results[i] == 1) {

player1\_wins++;

} else if (results[i] == 2) {

player2\_wins++;

} else {

draws++;

}

}

free(results);

printf("Chances of player 1 winning: %.2f%%\n", (double)player1\_wins / experiments \* 100);

printf("Chances of player 2 winning: %.2f%%\n", (double)player2\_wins / experiments \* 100);

printf("Chances of a draw: %.2f%%\n", (double)draws / experiments \* 100);

printf("Time taken: %.6f seconds\n", time\_taken);

return 0;

}

**Протокол работы программы**

**Тестирование:**

test.sh

#!/bin/bash

K=100

currentTurn=5

player1\_total=30

player2\_total=25

experiments=100000

threads=(1 4 7 10 13 16)

program="./out/lab\_2.out"

if [ ! -x "$program" ]; then

    echo "Исполняемый файл $program не найден или не является исполняемым."

    exit 1

fi

for num\_threads in "${threads[@]}"; do

    echo "Запуск программы с $num\_threads потоками..."

    $program $K $currentTurn $player1\_total $player2\_total $experiments $num\_threads

    echo "--------------------------------------------------"

done

-------------------------------------------------------

**bash test.sh**

**Запуск программы с 1 потоками...**

**Chances of player 1 winning: 49.33%**

**Chances of player 2 winning: 49.47%**

**Chances of a draw: 1.20%**

**Time taken: 0.394846 seconds**

**--------------------------------------------------**

**Запуск программы с 4 потоками...**

**Chances of player 1 winning: 49.73%**

**Chances of player 2 winning: 49.08%**

**Chances of a draw: 1.20%**

**Time taken: 0.138750 seconds**

**--------------------------------------------------**

**Запуск программы с 7 потоками...**

**Chances of player 1 winning: 49.38%**

**Chances of player 2 winning: 49.49%**

**Chances of a draw: 1.12%**

**Time taken: 0.080026 seconds**

**--------------------------------------------------**

**Запуск программы с 10 потоками...**

**Chances of player 1 winning: 49.64%**

**Chances of player 2 winning: 49.25%**

**Chances of a draw: 1.10%**

**Time taken: 0.072726 seconds**

**--------------------------------------------------**

**Запуск программы с 13 потоками...**

**Chances of player 1 winning: 49.40%**

**Chances of player 2 winning: 49.36%**

**Chances of a draw: 1.24%**

**Time taken: 0.056111 seconds**

**--------------------------------------------------**

**Запуск программы с 16 потоками...**

**Chances of player 1 winning: 49.48%**

**Chances of player 2 winning: 49.35%**

**Chances of a draw: 1.17%**

**Time taken: 0.049432 seconds**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Потоки /Эксперементы** | **1** | **4** | **7** | **10** | **13** | **16** |
| **100000** | **0.394846** | **0.138750** | **0.080026** | **0.072726** | **0.056111** | **0.049432** |
| **10000** | **0.039307** | **0.013346** | **0.008195** | **0.009680** | **0.009612** | **0.007270** |
| **1000** | **0.004090** | **0.001524** | **0.001273** | **0.001753** | **0.001829** | **0.001959** |

Strace:

strace ./out/lab\_2.out 100 5 30 25 3 2

execve("./out/lab\_2.out", ["./out/lab\_2.out", "100", "5", "30", "25", "3", "2"], 0x7ffe00f4a150 /\* 27 vars \*/) = 0

brk(NULL) = 0x562d0e9ee000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f3be79bc000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=20335, ...}) = 0

mmap(NULL, 20335, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f3be79b7000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f3be77a5000

mmap(0x7f3be77cd000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f3be77cd000

mmap(0x7f3be7955000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7f3be7955000

mmap(0x7f3be79a4000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7f3be79a4000

mmap(0x7f3be79aa000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f3be79aa000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f3be77a2000

arch\_prctl(ARCH\_SET\_FS, 0x7f3be77a2740) = 0

set\_tid\_address(0x7f3be77a2a10) = 3844

set\_robust\_list(0x7f3be77a2a20, 24) = 0

rseq(0x7f3be77a3060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f3be79a4000, 16384, PROT\_READ) = 0

mprotect(0x562ce2969000, 4096, PROT\_READ) = 0

mprotect(0x7f3be79f4000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7f3be79b7000, 20335) = 0

getrandom("\x01\xae\x69\x0d\x8f\x91\x76\x2f", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x562d0e9ee000

brk(0x562d0ea0f000) = 0x562d0ea0f000

rt\_sigaction(SIGRT\_1, {sa\_handler=0x7f3be783e520, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_ONSTACK|SA\_RESTART|SA\_SIGINFO, sa\_restorer=0x7f3be77ea320}, NULL, 8) = 0

rt\_sigprocmask(SIG\_UNBLOCK, [RTMIN RT\_1], NULL, 8) = 0

mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0) = 0x7f3be6fa1000

mprotect(0x7f3be6fa2000, 8388608, PROT\_READ|PROT\_WRITE) = 0

rt\_sigprocmask(SIG\_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f3be77a1990, parent\_tid=0x7f3be77a1990, exit\_signal=0, stack=0x7f3be6fa1000, stack\_size=0x7fff80, tls=0x7f3be77a16c0} => {parent\_tid=[3845]}, 88) = 3845

rt\_sigprocmask(SIG\_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT\_NONE, MAP\_PRIVATE|MAP\_ANONYMOUS|MAP\_STACK, -1, 0) = 0x7f3be67a0000

mprotect(0x7f3be67a1000, 8388608, PROT\_READ|PROT\_WRITE) = 0

rt\_sigprocmask(SIG\_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE\_VM|CLONE\_FS|CLONE\_FILES|CLONE\_SIGHAND|CLONE\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_CLEARTID, child\_tid=0x7f3be6fa0990, parent\_tid=0x7f3be6fa0990, exit\_signal=0, stack=0x7f3be67a0000, stack\_size=0x7fff80, tls=0x7f3be6fa06c0} => {parent\_tid=[3846]}, 88) = 3846

rt\_sigprocmask(SIG\_SETMASK, [], NULL, 8) = 0

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

write(1, "Chances of player 1 winning: 66."..., 36Chances of player 1 winning: 66.67%

) = 36

write(1, "Chances of player 2 winning: 33."..., 36Chances of player 2 winning: 33.33%

) = 36

write(1, "Chances of a draw: 0.00%\n", 25Chances of a draw: 0.00%

) = 25

write(1, "Time taken: 0.003578 seconds\n", 29Time taken: 0.003578 seconds

) = 29

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В рамках лабораторной работы была разработана многопоточная программа на языке C, которая моделирует игру в кости и определяет шансы на победу каждого из игроков на основе экспериментальных данных. Программа использует POSIX Threads (pthreads) для параллельного выполнения симуляций игр и потокобезопасный массив для хранения результатов. Программа также включает таймер для измерения времени выполнения.