Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Группа: М80-206Б-20

Студент: Сафуанов Р.А.

Преподаватель: Соколов А.А.

Москва, 2022

**Постановка задачи**

**Вариант 14.**

Составить программу на языке C, обрабатывающую данные в многопоточном режиме. При обработке использовать стандартные средства создания потоков операционной системы Unix. Ограничение потоков должно быть задано пользователем. Также необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы. В отчёте привести исследование зависимости ускорения и эффективности алгоритма от входящих данных и количества потоков. Получившиеся результаты необходимо объяснить.

Есть колода из 52 карт, рассчитать экспериментально (метод Монте-Карло) вероятность того, что сверху лежат две одинаковых карты. Количество раундов подаётся с ключом.

**Общий метод и алгоритм решения**

Использованные при реализации программы функции:

* int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \*attr); – инициализирует мьютекс с атрибутами, заданными attr. Если значение attr равно NULL, то используются атрибуты мьютекса по умолчанию.
* int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*func)(void \*), void \*arg); – запускает новый поток в вызывающем процессе. Новый поток начинает выполнение, вызывая func(); arg передаётся в качестве единственного аргумента func().
* int pthread\_join(pthread\_t thread, void \*\*retval); – ожидает завершения указанного потока. Если этот поток уже завершён, то функция pthread\_join() немедленно возвращается.
* int pthread\_mutex\_destroy (pthread\_mutex\_t \*mutex); – уничтожает мьютекс; мьютекс становится, по сути, неинициализированным.

Функция для потока получает на вход единственный аргумент – число раундов. В ней создаётся массив целых чисел размера 52 (число карт в колоде) и заполняется числами от 0 до 12, где 0 соответствует карте «2», 12 – карте «Туз». Каждое число в массиве будет повторяться 4 раза, аналогично количеству мастей в стандартной колоде.

При помощи функции rand() массив перемешивается, две крайние карты сравниваются и, если результат положительный – увеличиваем глобальную переменную-счётчик на 1 (для этого временно занимаем мьютекс).

**Код программы**

**main.c**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

const int DECK\_SIZE = 52;

int globalCount = 0;

pthread\_mutex\_t mutex;

void SwapInts(int \*a, int \*b) {

int tmp = \*a;

\*a = \*b;

\*b = tmp;

}

double globalTimeForRand = 0;

void \*FuncForThread(void \*arg) {

int localCount = 0;

int roundsNumber = \*((int \*)arg);

int deck[DECK\_SIZE];

int i, j, k;

double threadTimeForRand = 0;

for (k = 0; k < roundsNumber; ++k) {

for (i = 0; i < DECK\_SIZE; ++i) {

deck[i] = (i + 1) % 13; // cards value from 2 to 14 (or from 0 to 12)

}

for (i = DECK\_SIZE - 1; i >= 0; --i) {

struct timespec firstRandTime, lastRandTime;

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

j = rand() % (i + 1);

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

double localTimeForRand = (lastRandTime.tv\_sec - firstRandTime.tv\_sec);

localTimeForRand += (lastRandTime.tv\_nsec - firstRandTime.tv\_nsec) / 1000000000.0;

SwapInts(&deck[i], &deck[j]);

threadTimeForRand += localTimeForRand;

}

if (deck[0] == deck[1]) {

++localCount;

}

}

globalTimeForRand += threadTimeForRand;

pthread\_mutex\_lock(&mutex);

globalCount += localCount;

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

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

if (argc < 3) {

printf("Bad input, use ./a.out \*number of rounds\* \*number of threads\*\n");

return 0;

}

int roundsNumber = atoi(argv[1]);

int threadsNumber = atoi(argv[2]);

struct timespec firstTime, lastTime;

pthread\_t threadsArray[threadsNumber];

// Initializing mutex

if (pthread\_mutex\_init(&mutex, NULL) < 0) {

perror("pthread\_mutex\_init");

exit(EXIT\_FAILURE);

}

// Fix starting time

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

// Launching threads

int roundsPerThread = roundsNumber / threadsNumber;

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

if (pthread\_create(&threadsArray[i], NULL, FuncForThread, &roundsPerThread) != 0) {

perror("pthread\_create");

exit(EXIT\_FAILURE);

}

}

// Destroying threads

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

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

perror("pthread\_join");

exit(EXIT\_FAILURE);

}

}

// Fix ending time

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

double allTime = (lastTime.tv\_sec - firstTime.tv\_sec);

allTime += (lastTime.tv\_nsec - firstTime.tv\_nsec) / 1000000000.0;

// Destroying mutex

if (pthread\_mutex\_destroy(&mutex) < 0) {

perror("pthread\_mutex\_destroy");

exit(EXIT\_FAILURE);

}

printf("Probability is %.3lf\n", (double)globalCount / roundsNumber);

printf("Counted for %lf seconds\n", allTime);

printf("Time for rand() is %lf seconds\n", globalTimeForRand);

printf("Average time for rand() is %lf per thread\n", globalTimeForRand / threadsNumber);

printf("Time without rand is %lf\n", allTime - globalTimeForRand / threadsNumber);

fflush(stdout);

return 0;

}

**Тесты**

$ ./a.out 1000000 1

Probability is 0.059

Counted for 5.908855 seconds

Time for rand() is 2.999429 seconds

Average time for rand() is 2.999429 per thread

Time without rand is 2.909426

$ ./a.out 1000000 2

Probability is 0.059

Counted for 9.850075 seconds

Time for rand() is 16.719254 seconds

Average time for rand() is 8.359627 per thread

Time without rand is 1.490448

$ ./a.out 1000000 3

Probability is 0.059

Counted for 8.639979 seconds

Time for rand() is 22.737478 seconds

Average time for rand() is 7.579159 per thread

Time without rand is 1.060819

$ ./a.out 1000000 4

Probability is 0.058

Counted for 9.682424 seconds

Time for rand() is 35.374799 seconds

Average time for rand() is 8.843700 per thread

Time without rand is 0.838724

$ ./a.out 1000000 5

Probability is 0.059

Counted for 10.857552 seconds

Time for rand() is 50.594554 seconds

Average time for rand() is 10.118911 per thread

Time without rand is 0.738641

$ ./a.out 1000000 6

Probability is 0.059

Counted for 10.289037 seconds

Time for rand() is 57.935344 seconds

Average time for rand() is 9.655891 per thread

Time without rand is 0.633146

**Вывод**

При выполнении лабораторной работы, я научился работать с потоками в операционных системах семейства Unix, составил программу для многопоточного вычисления вероятности некоторого события (в нашем случае шанс появления двух одинаковых карт на вершине колоды). Программа оказалась относительно простой, а я на самом деле удивился тому, как просто порой можно рассчитывать вероятность некоторых событий на компьютере.

Также мне удалось поработать с мьютексами: без них практически никогда невозможно обойтись при разработке многопоточной программы.

Современные программы для оптимизации работают в многопоточном режиме, поэтому понимание базовых процессов работы с потоками пригодится в будущем.