Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Управление потоками в ОС**

Студент: Комбаров Владислав Александрович

Группа: М8О–212Б–22

Вариант: 15

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

## Целью является приобретение практических навыков в: • Управлении потоками в ОС • Обеспечении синхронизации между потоками

## Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке 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.

**Вариант 15:** Есть колода из 52 карт, рассчитать экспериментально (метод Монте-Карло) вероятность того, что сверху лежат две одинаковых карты. Количество раундов задаётся ключом программы

**Общие сведения о программе**

Программа lab2.c компилируется командой в терминале:

gcc -pthread lab2.c -o lab2

В программе используются следующие системные вызовы:

1. **pthread\_create** - создание нового потока.
2. **pthread\_join** - присоединение к основному потоку после завершения работы.
3. **pthread\_exit** - завершение работы потока с указанным кодом.

Программа экспериментально вычисляет вероятность того, что в колоде из 52 карт две самые верхние карты одинаковые. Вычисления проводятся на основе метода Монте-Карло. Этот метод заключается в том, чтобы провести указанное количество раундов, где колода карт будет перемешана случайным образом, затем отобрать случаи, которые соответствуют условию задачи, и разделить количество таких случаев на общее количество раундов. Ключом программы задаются два параметра: первый – это максимальное количество потоков, которое может работать одновременно; второй – общее количество раундов. За перемешивание колоды карт отвечает генератор случайных чисел, который выдаёт число от 0 до 51. В случае со второй картой мы используем цикл do while, чтобы исключить генерацию двух одинаковых чисел (все карты в колоде уникальные). Проверка на выполнение условия задачи состоит в равенстве остатков от деления на 13, так как одинаковых карт в колоде по 4 штуки (52 делить на 4 = 13). Каждый поток в программе обрабатывает свой диапазон раундов (первый поток от одного до частного от количества раундов на потоки, второй – от частного + 1 до частного \* 2 и т.д. Последний поток обрабатывает ещё остаток от этого частного.

**Общий метод и алгоритм решения**.

Для реализации задачи необходимо:

1. Придумать способ проведения эксперимента методом Монте-Карло
2. Написать функцию, обрабатывающую данные в одном потоке
3. Реализовать чтение ключа, задающего кол-во потоков и количество раундов, реализовать запуск потоков, присоединение их к основному, внедрить мьютекс в программу и своевременно его уничтожить.
4. Засечь время выполнения программы, подсчитать его, а также подсчитать искомую вероятность.
5. Проанализировать зависимость времени работы программы от количества используемых потоков.

**Основные файлы программы**

**lab2.c:**

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <time.h>

int round\_count; // Количество раундов

int successes = 0; // Количество успешных раундов

int max\_threads; // Максимальное количество потоков

pthread\_mutex\_t mutex; // Мьютекс для обеспечения синхронизации

// Функция для обработки данных в потоке

void\* process\_data(void\* arg) {

int thread\_id = \*(int\*)arg;

//printf("%d\n", thread\_id);

int rounds\_per\_thread = round\_count / max\_threads; // Количество раундов для каждого потока

int remaining\_rounds = 0;

int a[round\_count];

int b[round\_count];

for (int i = 0; i < round\_count; i++) a[i] = 52;

for (int i = 0; i < round\_count; i++) b[i] = 52;

// Распределение оставшихся раундов на последний поток

if (thread\_id == max\_threads - 1) {

remaining\_rounds = round\_count % max\_threads; // Оставшиеся раунды после деления по равным частям

}

// Выполнение эксперимента для каждого потока

for (int i = thread\_id\*rounds\_per\_thread; i < rounds\_per\_thread\*(thread\_id + 1) + remaining\_rounds; i++) {

// Перемешивание колоды карт

do {a[i] = rand() % 52;

b[i] = rand() % 52;} while (a[i] == b[i]);

}

/\*for(int i = 0; i < round\_count; i++) printf("%d ",a[i]);

printf("\n");

for(int i = 0; i < round\_count; i++) printf("%d ",b[i]);

printf("\n");\*/

for (int i = thread\_id\*rounds\_per\_thread; i < rounds\_per\_thread\*(thread\_id + 1) + remaining\_rounds; i++) {

// Проверка наличия двух одинаковых карт сверху

if (a[i] % 13 == b[i] % 13) {

pthread\_mutex\_lock(&mutex);

successes++;

pthread\_mutex\_unlock(&mutex);

}

}

pthread\_exit(NULL);

}

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

if (argc != 3) {

printf("Необходимо задать ограничение на количество потоков и количество раундов\n");

return -1;

}

max\_threads = atoi(argv[1]);

round\_count = atoi(argv[2]);

pthread\_t threads[max\_threads];

int thread\_ids[max\_threads];

// Инициализация мьютекса

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

printf("Ошибка при инициализации мьютекса\n");

return -1;

}

// Инициализация генератора случайных чисел

srand(time(NULL));

clock\_t start\_time = clock();

// Создание и запуск потоков

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

thread\_ids[i] = i;

pthread\_create(&threads[i], NULL, process\_data, &thread\_ids[i]);

}

// Ожидание завершения всех потоков

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

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

}

clock\_t end\_time = clock();

// Уничтожение мьютекса

pthread\_mutex\_destroy(&mutex);

// Расчет и вывод экспериментальной вероятности

double experimental\_probability = (double) successes / round\_count;

printf("Экспериментальная вероятность: %f\n", experimental\_probability);

double execution\_time = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC; // Подсчет времени в секундах

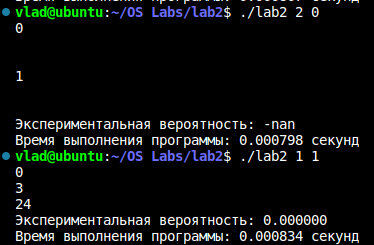
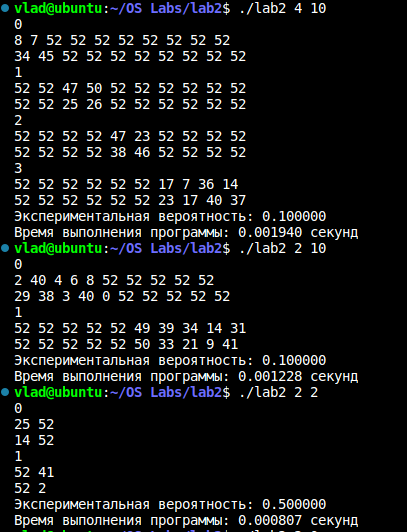
printf("Время выполнения программы: %lf секунд\n", execution\_time);

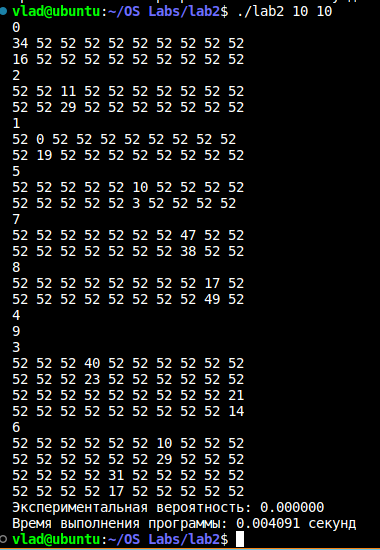
return 0;

}

**Пример работы**

Для наглядности на экран были выведены идентификатор потока и затем – массив с результатами эксперимента (a – массив первых карт, b – массив вторых карт). Число 52 означает, что данный поток этот диапазон не обрабатывал.





**Таблицы зависимости времени работы от количества потоков:**

При 10 раундах:

|  |  |
| --- | --- |
| Кол-во потоков | Время работы |
| 1 | 0.015 секунд |
| 2 | 0.008 секунд |
| 5 | 0.005 секунд |
| 7 | 0.003 секунд |
| 10 | 0.0066 секунд |

При 1 000 000 раундах

|  |  |
| --- | --- |
| Кол-во потоков | Время работы |
| 1 | 0.14 секунд |
| 2 | 0.48 секунд |
| 5 | 0.35 секунд |
| 7 | 0.42 секунд |
| 10 | 0.51 секунд |

**Вывод**

В процессе работы я получил навыки работы с многопоточностью в ОС. Основными ошибками при выполнении лабораторной работы стали: непонимание принципа многопоточности – первоначально программа была построена по схеме «один поток производит вычисления, присоединяется к основному и только после этого разрешает подключиться другому». Переосмыслив суть данного явления, программа была переделана под обработку данных несколькими потоками одновременно. Это видно из скриншотов с тестами, где выводится два подряд идентификатора потока и только потом массивы, которые они обрабатывают. Также был риск сравнения двух случайных чисел, одно из которых сгенерировано одним потоком, а другое – другим, в то время как потоки разбивают раунды, но не пары чисел. Из-за этого пришлось использовать массивы, с ячейками которых связан какой-либо раунд (например, a[0] и b[0] – результаты первого раунда). Помимо этого, нужно было разобраться с применением мьютексов: когда они замедляют работу программы, а когда ускоряют. Менее глобальными ошибками были не идентифицированный генератор случайных чисел и изначально не был учтён случай a[i] = b[i] – случай полного совпадения двух карт в колоде, чего по условию быть не может.

После проведения замеров времени я пришел к выводу, что основной прирост в производительности происходит при использовании 7-8 потоков. Дальнейшее увеличение числа потоков не дает серьезного прироста в производительности, время выполнения программы начинает увеличиваться. Но это только при проведении 10 раундов. При проведении большого количества раундов (я взял один миллион) происходит ситуация, когда использование одного потока затрачивает меньше времени, чем использование двух. Возможно, дело в самой структуре программы: каждый поток должен создать массив размером один миллион и заполнить его числами “52” и только затем изменить в массиве свой диапазон и сверить, является ли раунд успешным или нет. Повлиять может и блокировка мьютекса во время изменения глобальной переменной successes и последующая его разблокировка. Я получил в своей программе, что многопоточность не всегда может хорошо влиять на производительность программы. Всё зависит от количества обрабатываемых данных и самой структуры написанной программы.