Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Операционные системы" №2

Студент: Брюханов З. Д.

Преподаватель: Миронов Е.С.

Группа: М8О-207Б-22

Дата: 29.09.2023

Оценка:

Подпись:

Москва 2023 г.

Оглавление

Цель работы

Целью является приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

Постановка задачи

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.

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 алгоритм решения

Запрашиваем у пользователя количество потоков, на которых должна выполняться программа. Если этих потоков больше чем максимальное число потоков предоставляемое системой, устанавливаем это максимальное количество, потому что от большего количества потоков не будет смысла, в противном случае устанавливаем столько потоков, сколько запросил пользователь. Считываем с консоли все данные требуемые для вычисления программы. Затем создаем структуру **thread_data**, которая содержит данные

необходимые для работы каждого отдельного потока. Для последнего потока устанавливаем количество экспериментов всех до конца, это необходимо чтобы если количество экспериментов не поделилось нацело на количество потоков, то программа все равно проделала все запрошенные эксперименты. После завершения работы потока мы извлекаем из его массива с ответами результаты в общий массив с ответами, так как это делается в блоке где используется **pthread_join**, то поток там уже один и их не надо синхронизировать, вследствие чего мне не пришлось использовать семафоры или мьютексы, потому что каждый поток работал только со своими данными. Когда результаты работы всех потоков были обработаны, результат их вычислений выводится на экран. Также я замеряю время вычислений на потоках, чтобы было удобнее сравнивать скорость работы при запуске с различным числом потоков.

Реализация

main.c

```
#include <pthread.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>
const int MAX_SIZE_OF_STRING = 50;
const int SIZE_OF_RESULT_ARRAY = 3;
typedef struct {
    int number_of_remaining_throws_in_round;
    int first_score;
    int second_score;
    int number_of_experiments;
    int* result_array;
} thread_data;
void error_processing(bool exception, char* bug_report) {
    if (exception) {
        write(STDERR_FILENO, bug_report, strlen(bug_report) * sizeof(char));
        exit(STDERR_FILENO);
    }
}
void limiting_the_number_of_threads(int* number_of_threads) {
```

```
    int maximum_number_of_threads_in_the_system = (int) sysconf(_SC_NPROCESSORS_ONLN);
    if (*number_of_threads > maximum_number_of_threads_in_the_system) {
        *number_of_threads = maximum_number_of_threads_in_the_system;
        char maximum_number_of_threads_in_the_system_string[MAX_SIZE_OF_STRING];
        sprintf(maximum_number_of_threads_in_the_system_string, "%d",
maximum_number_of_threads_in_the_system);
        write(STDOUT_FILENO,
              "the limit of threads in the system has been exceeded, the number of threads has been set: ",
              strlen("the limit of threads in the system has been exceeded, the number of threads has been
set: "));
        write(STDOUT_FILENO, maximum_number_of_threads_in_the_system_string,
              strlen(maximum_number_of_threads_in_the_system_string));
        write(STDOUT_FILENO, "\n", 1);
    }
}
void input_string_correction(char* string) {
    for (unsigned long symbol_index = 0; symbol_index < strlen(string); ++symbol_index) {
        if (string[symbol_index] == '\n') {
            string[symbol_index] = '\0';
            return;
        }
    }
}
int read_int(const char* message_to_the_user) {
    char input_buffer[MAX_SIZE_OF_STRING];
    if (strlen(message_to_the_user) > 0) {
        write(STDOUT_FILENO, message_to_the_user, strlen(message_to_the_user));
    }
    error_processing((read(STDIN_FILENO, input_buffer, sizeof(input_buffer)) <= 0), "console
reading error");
    input_string_correction(input_buffer);
    return atoi(input_buffer);
}
void output_information_as_a_result_of_program(int* result,
                                               const int64_t time_of_program,
                                               const int number_of_threads,
                                               const int number_of_experiments) {
    const float chance_of_winning_the_first_player = (float) result[1] / (float)
number_of_experiments;
    const float chance_of_winning_the_second_player = (float) result[2] / (float)
number_of_experiments;
    char time_of_program_string[MAX_SIZE_OF_STRING];
    char number_of_threads_string[MAX_SIZE_OF_STRING];
    char result_first_player[MAX_SIZE_OF_STRING];
    char result_second_player[MAX_SIZE_OF_STRING];
    char result_draw[MAX_SIZE_OF_STRING];
    char chance_of_winning_the_first_player_string[MAX_SIZE_OF_STRING];
    char chance_of_winning_the_second_player_string[MAX_SIZE_OF_STRING];
```

```
    sprintf(time_of_program_string, "%lld", time_of_program);
    sprintf(number_of_threads_string, "%d", number_of_threads);
    sprintf(result_first_player, "%d", result[1]);
    sprintf(result_second_player, "%d", result[2]);
    sprintf(result_draw, "%d", result[0]);
    sprintf(chance_of_winning_the_first_player_string, "%f", chance_of_winning_the_first_player);
    sprintf(chance_of_winning_the_second_player_string, "%f",
chance_of_winning_the_second_player);
    write(STDOUT_FILENO, "Number of thread = ", strlen("number of thread = "));
    write(STDOUT_FILENO, number_of_threads_string, strlen(number_of_threads_string));
    write(STDOUT_FILENO, "\n", 1);
    write(STDOUT_FILENO, "time spent on the game = ", strlen("time spent on the game = "));
    write(STDOUT_FILENO, time_of_program_string, strlen(time_of_program_string));
    write(STDOUT_FILENO, "\n", 1);
    write(STDOUT_FILENO, "The result is a draw: ", strlen("The result is a draw: "));
    write(STDOUT_FILENO, result_draw, strlen(result_draw));
    write(STDOUT_FILENO, "\n", 1);
    write(STDOUT_FILENO, "The result won first: ", strlen("The result won first: "));
    write(STDOUT_FILENO, result_first_player, strlen(result_first_player));
    write(STDOUT_FILENO, "\n", 1);
    write(STDOUT_FILENO, "The result won second: ", strlen("The result won second: "));
    write(STDOUT_FILENO, result_second_player, strlen(result_second_player));
    write(STDOUT_FILENO, "\n\n", 2);
    write(STDOUT_FILENO, "The chance of winning the first player: ",
          strlen("The chance of winning the first player: "));
    write(STDOUT_FILENO, chance_of_winning_the_first_player_string,
strlen(chance_of_winning_the_first_player_string));
    write(STDOUT_FILENO, "\n", 1);
    write(STDOUT_FILENO, "The chance of winning the second player: ",
          strlen("The chance of winning the second player: "));
    write(STDOUT_FILENO, chance_of_winning_the_second_player_string,
          strlen(chance_of_winning_the_second_player_string));
    write(STDOUT_FILENO, "\n", 1);
}
int one_game(const int number_of_remaining_throws_in_round, int first_score, int second_score) {
    for (int tour_number = 0; tour_number < number_of_remaining_throws_in_round; +
+tour_number) {
        first_score += rand() % 6 + 1 + rand() % 6 + 1;
        second_score += rand() % 6 + 1 + rand() % 6 + 1;
    }
    if (first_score > second_score) {
        return 1;
    } else if (first_score < second_score) {
        return 2;
    } else {
        return 0;
    }
}
```

```
void* thread_function(void* parameters) {
    thread_data* data = (thread_data*) parameters;
    for (int index_of_experiments = 0; index_of_experiments < data->number_of_experiments; +
+index_of_experiments) {
        int winner = one_game(data->number_of_remaining_throws_in_round, data->first_score, data-
>second_score);
        if (winner == 1) {
            data->result_array[1] += 1;
        } else if (winner == 2) {
            data->result_array[2] += 1;
        } else {
            data->result_array[0] += 1;
        }
    }
    pthread_exit(NULL);
}
void starting_and_configuring_threads(const int number_of_threads_from_the_user,
                                      const int number_of_throws_in_tour,
                                      const int tour_number,
                                      const int number_points_of_first_player,
                                      const int number_points_of_second_player,
                                      const int number_of_experiments,
                                      int* result) {
    int experiments_per_thread = number_of_experiments / number_of_threads_from_the_user;
    pthread_t* threads_array = (pthread_t*) malloc(number_of_threads_from_the_user *
sizeof(pthread_t));
    thread_data* array_thread_data = (thread_data*) malloc(number_of_threads_from_the_user *
sizeof(thread_data));
    for (int index_of_array_thread = 0;
         index_of_array_thread < number_of_threads_from_the_user;
         ++index_of_array_thread) {
        array_thread_data[index_of_array_thread].number_of_remaining_throws_in_round =
                number_of_throws_in_tour - tour_number;
        array_thread_data[index_of_array_thread].first_score = number_points_of_first_player;
        array_thread_data[index_of_array_thread].second_score = number_points_of_second_player;
        array_thread_data[index_of_array_thread].number_of_experiments = experiments_per_thread;
        array_thread_data[index_of_array_thread].result_array = malloc(SIZE_OF_RESULT_ARRAY
* sizeof(int));
        if (index_of_array_thread == number_of_threads_from_the_user - 1) {
            array_thread_data[index_of_array_thread].number_of_experiments +=
                    number_of_experiments % number_of_threads_from_the_user;
        }
        error_processing((pthread_create(&(threads_array[index_of_array_thread]), NULL,
thread_function,
                                         &array_thread_data[index_of_array_thread])) != 0, "Error creating
thread");
    }
    for (int index_of_array_thread = 0;
```

```
        index_of_array_thread < number_of_threads_from_the_user;
        ++index_of_array_thread) {
       error_processing(((threads_array[index_of_array_thread], NULL)) != 0, "Thread termination
error");
       for (int index_of_result_array = 0; index_of_result_array < SIZE_OF_RESULT_ARRAY; +
+index_of_result_array) {
          result[index_of_result_array] +=
array_thread_data[index_of_array_thread].result_array[index_of_result_array];
       }
       free(array_thread_data[index_of_array_thread].result_array);
   }
   free(threads_array);
   free(array_thread_data);
}
int64_t current_time_millis() {
   struct timeval time;
   gettimeofday(&time, NULL);
   return (int64_t) (time.tv_sec) * 1000 + (int64_t) (time.tv_usec) / 1000;
}
int main(int argc, char* argv[]) {
   error_processing(argc != 2, "write the number of threads");
   int number_of_threads_from_the_user = atoi(argv[1]);
   limiting_the_number_of_threads(&number_of_threads_from_the_user);
   srand((unsigned) time(NULL));
   int number_of_throws_in_tour = read_int("enter the number of throws: ");
   int tour_number = read_int("enter the tour number: ");
   int number_points_of_first_player = read_int("enter the number of points of the first player: ");
   int number_points_of_second_player = read_int("enter the number of points of the second player:
");
   int number_of_experiments = read_int("enter the number of experiments that the program should
perform: ");
   error_processing((number_of_throws_in_tour < tour_number),
                    "Incorrect data: the number of the round cannot be greater than the total number of
throws");
   int result[] = {0, 0, 0};
   int64_t start_time = current_time_millis();
   starting_and_configuring_threads(
           number_of_threads_from_the_user,
           number_of_throws_in_tour,
           tour_number,
           number_points_of_first_player,
           number_points_of_second_player,
           number_of_experiments,
           result);
   int64_t stop_time = current_time_millis();
   output_information_as_a_result_of_program(result,
                                             stop_time - start_time,
                                             number_of_threads_from_the_user,
```

```
                                    number_of_experiments);
    return 0;
}
```

Пример работы

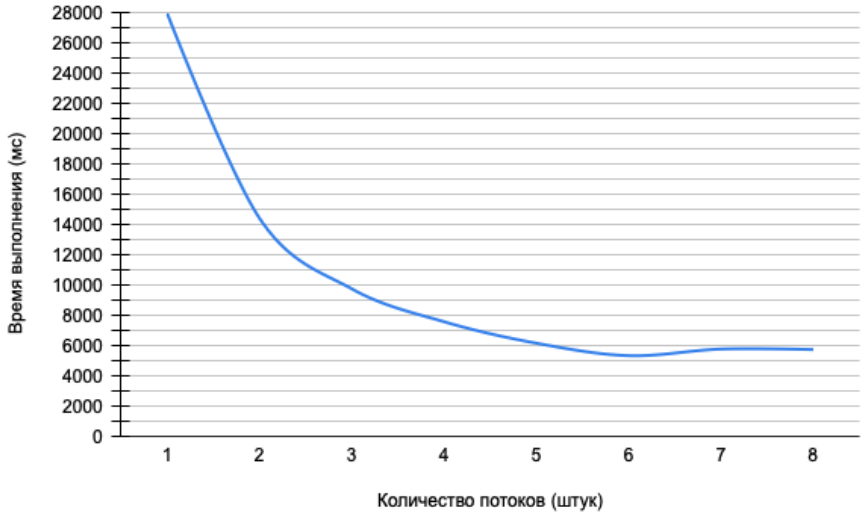
Test 1

```
br_zahar@MacBook-Pro-Zahar build % ./task02 8
enter the number of throws: 1000
enter the tour number: 2
enter the number of points of the first player: 12
enter the number of points of the second player: 19
enter the number of experiments that the program should perform: 1000000
Number of thread = 8
time spent on the game = 5732
The result is a draw: 3688
The result won first: 472395
The result won second: 523917

The chance of winning the first player: 0.472395
The chance of winning the second player: 0.523917
```

Test 2

```
br_zahar@MacBook-Pro-Zahar build % ./task02 1
enter the number of throws: 1000
enter the tour number: 2
enter the number of points of the first player: 12
enter the number of points of the second player: 19
enter the number of experiments that the program should perform: 1000000
Number of thread = 1
time spent on the game = 27926
The result is a draw: 3631
The result won first: 471993
The result won second: 524376

The chance of winning the first player: 0.471993
The chance of winning the second player: 0.524376
```

График скорости/количества потоков



Вывод

В ходе выполнения лабораторной работы, я познакомился с потоками и узнал, как с их помощью увеличить скорость выполнения программы. Стоит учитывать, что если создать потоков больше, чем ядер процессора, от этого практически не будет никакой пользы, так как количество потоков, которые могут работать одновременно не больше количества ядер. По графику видно, что программа быстрее всего работает при 6 потоков, это можно объяснить тем, что когда потоков больше чем 6, они начинают выполнятся не параллельно, а по очереди. То есть некоторое время поработает один поток, потом другой. При смене работающих потоков тратится время на их остановку и запуск, из-за чего не всегда чем больше потоков, тем быстрее работает программа.