Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу
«Операционные системы»

Тема работы
“Потоки”

Студент: Борисов Ян Артурович
Группа: М8О-208Б-20
Вариант: 7
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

Содержание

Репозиторий

https://github.com/Yannikupy/OS/tree/master/os_lab3

Постановка задачи

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение потоков должно быть задано ключом запуска вашей программы. Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

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

Общие сведения о программе

Программа представляет из себя один файл main.c.

Общий метод и алгоритм решения

Проводятся эксперименты с играми в кости. Если количество экспериментов, меньше чем заданное количество потоков, то мы выделяем поток на каждый эксперимент, если же больше чем количество потоков, то мы в цикле выделяем потоки на каждый эксперимент, дожидаемся их завершения и заново делаем то же самое, пока количество оставшихся экспериментов не будет <= количества потоков.

Исходный код

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdbool.h>

struct Arguments {
    int id; //  ID потока
```

```
    int num_of_throws;
    int sum_of_points_1;
    int sum_of_points_2;
    bool win_1;
    bool win_2;
};

typedef struct Arguments Args;



void* thread_func(void *args) {
    Args *arg = (Args*) args;
    int id = arg->id;
    int num_of_throws = arg->num_of_throws;
    int sum_of_points_1 = arg->sum_of_points_1;
    int sum_of_points_2 = arg->sum_of_points_2;
    int player_1_num_1;
    int player_1_num_2;
    int player_2_num_1;
    int player_2_num_2;

    for (int index = 0; index < num_of_throws; index++) {
        player_1_num_1 = rand() % 6 + 1;
        player_1_num_2 = rand() % 6 + 1;
        player_2_num_1 = rand() % 6 + 1;
        player_2_num_2 = rand() % 6 + 1;
        printf("ID of thread: %d\n", id);
        printf("For the 1st player: %d and %d\n", player_1_num_1,
player_1_num_2);
        printf("For the 2nd player: %d and %d\n", player_2_num_1,
player_2_num_2);
        sum_of_points_1 += (player_1_num_1 + player_1_num_2);
        sum_of_points_2 += (player_2_num_1 + player_2_num_2);
        printf("\n");
    }

    if (sum_of_points_1 > sum_of_points_2) {
        arg->win_1 = true;
        arg->win_2 = false;
    }
    else {
        arg->win_1 = false;
        arg->win_2 = true;
    }

    return NULL;
}

int main (int argc, char *argv[]) {
    if (argc != 7) {
        fprintf(stderr, "Usage: %s, Num_of_throws (K), Num_of_tour, Points_1,
Points_2, Num_of_experiments Max_num_of_threads\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    if (atoi(argv[1]) < 0 || atoi(argv[2]) < 0 || atoi(argv[3]) < 0 ||
atoi(argv[4]) < 0 || atoi(argv[5]) < 0 || atoi(argv[6]) < 0) {
        fprintf(stderr, "Arguments %d, %d, %d, %d, %d, %d must be non nega-
tive\n", atoi(argv[1]), atoi(argv[2]), atoi(argv[3]), atoi(argv[4]),
atoi(argv[5]), atoi(argv[6]));
        exit(EXIT_FAILURE);
    }
```

```
    int status;
    int status_addr;
    int num_of_throws = atoi(argv[1]) - atoi(argv[2]);
    int sum_of_points_1 = atoi(argv[3]);
    int sum_of_points_2 = atoi(argv[4]);
    int num_of_experiments = atoi(argv[5]);
    size_t num_of_threads = (size_t)atoi(argv[6]);
    bool need_continue = true;

    int num_of_plays = num_of_experiments;
    int wins_1 = 0;
    int wins_2 = 0;
    float result_for_the_1st;
    float result_for_the_2nd;

    srand(time(NULL));

    while(need_continue) {
        if (num_of_threads < num_of_experiments) {
            pthread_t *threads = (pthread_t *) calloc(num_of_threads,
sizeof(pthread_t));
            if (threads == NULL) {
                fprintf(stderr, "in main: Can't allocate memory for
threads\n");
                exit(EXIT_FAILURE);
            }
            //  вводим значения для аргументов
            Args args[num_of_threads];
            for (int index = 0; index < num_of_threads; index++) {
                args[index].id = index;
                args[index].num_of_throws = num_of_throws;
                args[index].sum_of_points_1 = sum_of_points_1;
                args[index].sum_of_points_2 = sum_of_points_2;
            }

            //  создаем новые потоки
            for (int index = 0; index < num_of_threads; index++) {
                status = pthread_create(&threads[index], NULL, thread_func,
(void *) &args[index]);
                if (status != 0) {
                    fprintf(stderr, "main error: Can't create thread, status
= %d\n", status);
                    exit(EXIT_FAILURE);
                }
            }

            //  ждем завершения работы потоков
            for (int index = 0; index < num_of_threads; index++) {
                status = pthread_join(threads[index], (void **) &sta-
tus_addr);
                if (status != 0) {
                    fprintf(stderr, "main error: Can't join thread, status =
%d\n", status);
                    exit(EXIT_FAILURE);
                }
            }

            //  рассчитываем количество побед у каждого игрока
            for (int index = 0; index < num_of_threads; index++) {
                if (args[index].win_1) {
                    ++wins_1;
                }
                else {
```

```
                    ++wins_2;
                }
            }

            num_of_experiments -= num_of_threads;
            free(threads);
        }
        else {
            pthread_t *threads = (pthread_t *) calloc(num_of_experiments,
sizeof(pthread_t));
            if (threads == NULL) {
                fprintf(stderr, "in main: Can't allocate memory for
threads\n");
                exit(EXIT_FAILURE);
            }
            //  вводим значения аргументов
            Args args[num_of_experiments];
            for (int index = 0; index < num_of_experiments; index++) {
                args[index].id = index;
                args[index].num_of_throws = num_of_throws;
                args[index].sum_of_points_1 = sum_of_points_1;
                args[index].sum_of_points_2 = sum_of_points_2;
            }

            //  создаем новые потоки
            for (int index = 0; index < num_of_experiments; index++) {
                status = pthread_create(&threads[index], NULL, thread_func,
(void *) &args[index]);
                if (status != 0) {
                    fprintf(stderr, "main error: Can't create thread, status
= %d\n", status);
                    exit(EXIT_FAILURE);
                }
            }

            //  ждем завершения работы потоков
            for (int index = 0; index < num_of_experiments; index++) {
                status = pthread_join(threads[index], (void **) &sta-
tus_addr);
                if (status != 0) {
                    fprintf(stderr, "main error: Can't join thread, status =
%d\n", status);
                    exit(EXIT_FAILURE);
                }
            }

            //  рассчитываем количество побед у каждого игрока
            for (int index = 0; index < num_of_experiments; index++) {
                if (args[index].win_1) {
                    ++wins_1;
                }
                else {
                    ++wins_2;
                }
            }

            free(threads);
            need_continue = false;
        }
    }

    //  подсчитываем шансы на победу каждого игрока
    result_for_the_1st = (float)wins_1 / (float)num_of_plays;
```

```
    result_for_the_2nd = 1.0 - result_for_the_1st;
    printf("Chances of the 1st player: %.2f\n", result_for_the_1st);
    printf("Chances of the 2nd player: %.2f\n", result_for_the_2nd);

    return 0;
}
```

Демонстрация работы программы

Ввод в консоль:

./a.out 5 2 4 7 3 6

ID of thread: 0

For the 1st player: 3 and 5

For the 2nd player: 3 and 4

ID of thread: 1

For the 1st player: 4 and 4

ID of thread: 0

For the 1st player: 4 and 3

For the 2nd player: 5 and 2

ID of thread: 0

For the 1st player: 5 and 3

For the 2nd player: 4 and 5

ID of thread: 2

For the 1st player: 5 and 4

For the 2nd player: 2 and 3

ID of thread: 2

For the 1st player: 6 and 5

For the 2nd player: 5 and 5

ID of thread: 2

For the 1st player: 3 and 6

For the 2nd player: 5 and 5

ID of thread: 1

For the 1st player: 5 and 1

For the 2nd player: 6 and 6

ID of thread: 1

For the 1st player: 5 and 6

For the 2nd player: 1 and 3

Chances of the 1st player: 0.33

Chances of the 2nd player: 0.67

Выводы

Проделав лабораторную работу, я приобрёл практические навыки в управлении потоками в ОС и обеспечил синхронизацию между ними, воспользовался утилитой strace.