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-216БВ-24

Студент: Попов К.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 18.11.25

Москва, 2025

Постановка задачи

Вариант 15.

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 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.

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

Общий метод и алгоритм решения

Ключевые функции:

- int pthread_create(pthread_t *restrict newthread, const pthread_attr_t *restrict attr, void *(*start_routine)(void *), void *restrict arg). Создаёт новый поток, начинающийся с выполнения функции START-ROUTINE, которая получает аргументы из переданного ARG. ATTR содержит атрибуты создания потока. Новый идентификатор потока сохраняется в *NEWTHREAD. Возвращает ноль в случае успеха.
- int pthread_join(pthread_t th, void **thread_return). Заставляет вызывающий поток ожидать завершения потока TH. Статус завершения потока сохраняется в *THREAD_RETURN, если значение THREAD_RETURN не равно NULL.
- int clock_gettime(clockid_t clock_id, struct timespec *tp). Получает текущее время из часов CLOCK_ID и записывает его в TP. В работе использовались часы CLOCK_MONOTONIC, которые отсчитывают время с определенной точки (обычно с момента загрузки системы) и не могут быть скорректированы назад (в отличие от CLOCK_REALTIME).
- int rand_r(unsigned int *seedp). Генерирует и возвращает псевдослучайное число от 0 до RAND_MAX. Является потокобезопасной, в отличие от rand(), так как принимает указатель *SEEDP на состояние генератора случайных чисел, определяемое отдельно для каждого потока.
- ssize_t write(int fd, const void *buf, size_t n) - пишет N байт из BUF в дескриптор FD. Возвращает количество записанных байт или -1 в случае ошибки.

Игральные карты имеют два параметра: ранг (13 вариантов) и масть (4 варианта). В колоде из 52 карт все карты различны по комбинациям этих двух параметров. Будем считать игральные карты одинаковыми, если их ранги одинаковы.

Метод Монте-Карло заключается в проведении большого количества экспериментов со случайными величинами. В контексте задачи для определения вероятности в качестве эксперимента будет служить алгоритм, который случайным образом перемешивает карты в колоде и затем сравнивает две последние (верхние) карты. После некоторого количества экспериментов приблизительная искомая вероятность будет равна частному от деления числа исходов,

удовлетворяющих событию “две верхние карты равны”, на число всех исходов. При увеличении числа экспериментов точность определения вероятности увеличивается.

В параллельной реализации алгоритма каждый эксперимент выполняется некоторым потоком (библиотека pthread). Количество потоков положим как минимальное число из максимально допустимого количества потоков и заданного общего числа экспериментов. Каждому потоку сообщается (путем передачи аргумента в исполняемую функцию) число экспериментов, которое он должен промоделировать, все эксперименты распределяются между потоками равномерно до запуска алгоритма. Результат серии экспериментов - количество исходов с одинаковыми картами - каждый поток записывает в свою ячейку памяти. После завершения всех потоков результаты суммируются и полученное число делится на общее число экспериментов.

Параллельная реализация будет вести себя так же, как и последовательная, при запуске на одном потоке.

Для более точных измерений скомпилированная программа запускает алгоритм TEST_COUNT раз, а затем подсчитывает среднее значение времени исполнения, а также искомой вероятности.

В таблице 1 приведены результаты вычисления вероятности методом Монте-Карло, а также время исполнения, ускорение и эффективность в зависимости от числа экспериментов для 16 потоков на 16 логических ядрах по сравнению с одним потоком.

Таблица 1 – Результаты вычисленной вероятности и измерений для разного числа экспериментов

Число экспериментов	**Вычисленная вероятность**	**Время исполнения на 16 потоках (мс)**	**Время исполнения на одном потоке (мс)**	**Ускорение**	**Эффективность**
10	0,1	3,265	0,358	0,091	0,011
100	0,05	3,58	0,353	0,218	0,027
1000	0,063	3,933	1,081	0,686	0,086
10000	0,0577	3,405	7,148	2,762	0,345
100000	0,0578	18,254	63,302	3,817	0,477
1000000	0,058796	78,864	622,303	6,550	0,819
10000000	0,05879	548,065	6211,388	7,024	0,878

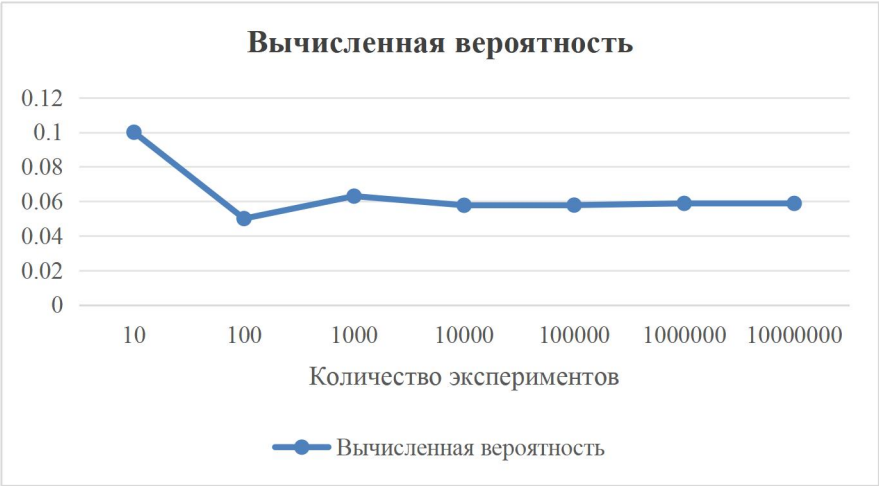

Рисунок 1 – Зависимость вероятности по Монте-Карло от входных данных

Теоретическая вероятность

Первая карта сверху может быть любой. После взятия первой карты в колоде остается 51 карта. Карт того же ранга, что и первая, остается 3. Значит, искомая вероятность равна 3/51 ≈ 0.058824. Вероятность по Монте-Карло стремится к этому значению.

На рисунках 2, 3 и 4 визуализированы результаты измерений.

Ускорение показывает, во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с последовательным алгоритмом. Ускорение определяется величиной $S_N = T_1 / T_N$, где T_1 – время выполнения на одном потоке, T_N – время выполнения на N потоках.

Эффективность – величина $E_N = S_N / N$, где S_N – ускорение, N – количество используемых потоков.

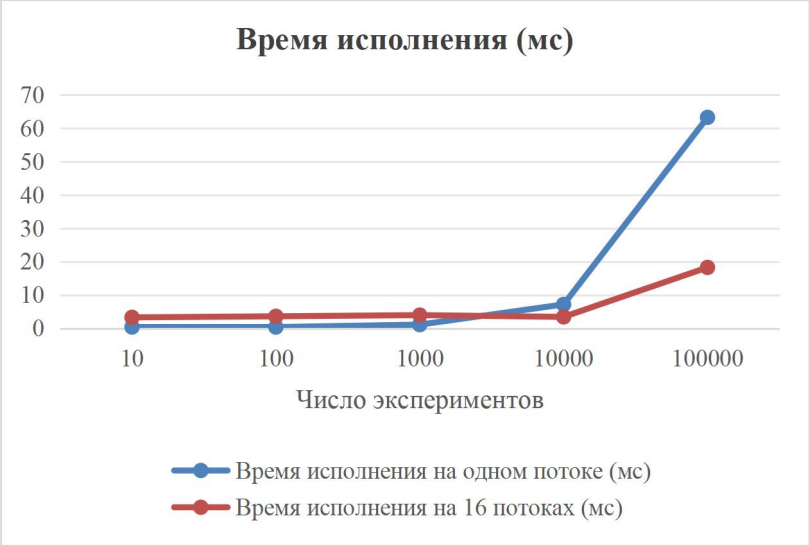

Рисунок 2 – Зависимость времени исполнения от входных данных при разном количестве потоков

Рисунок 3 – Зависимость ускорения от входных данных для 16 потоков

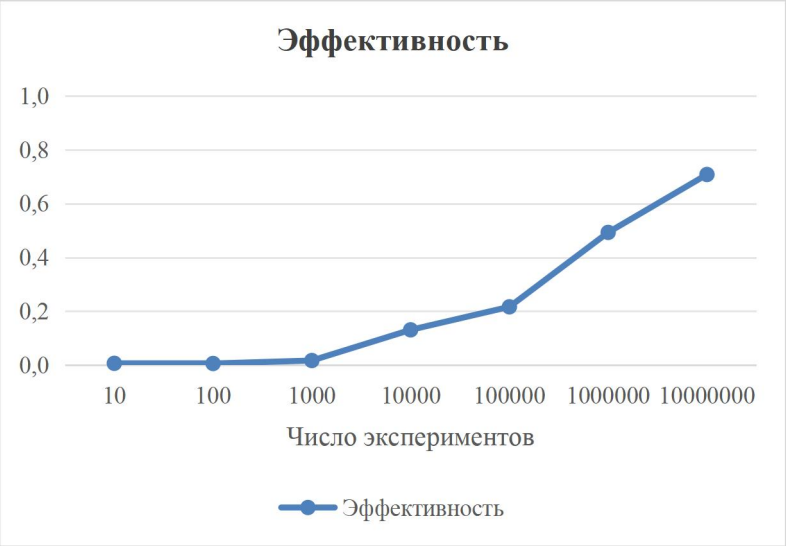

Рисунок 4 – Зависимость эффективности от входных данных для 16 потоков

Из результатов измерений видно, что многопоточность эффективна только для достаточно больших объемов вычислений, иначе накладные расходы на поддержку нескольких потоков превышают экономию времени.

В таблице 2 приведены результаты измерений времени исполнения, ускорения и эффективности для 10 млн экспериментов и максимального числа логических ядер 16. На рисунках 5, 6 и 7 данные визуализированы в виде графиков.

Таблица 2 – Результаты измерений для разного количества потоков

Число потоков	**Время исполнения (мс)**	**Ускорение**	**Эффективность**
1	3215,118	1,00	1,00
2	1716,931	1,87	0,94
3	1205,020	2,67	0,89
4	920,695	3,49	0,87
5	745,200	4,31	0,86
6	643,813	4,99	0,83
8	505,850	6,36	0,79
12	356,342	9,02	0,75
15	302,116	10,64	0,71
16	299,131	10,75	0,67
32	302,348	10,63	0,33
64	297,994	10,79	0,17
128	296,907	10,83	0,08
512	303,822	10,58	0,02
1024	326,676	9,84	0,01
10000	1073,979	2,99	0,00

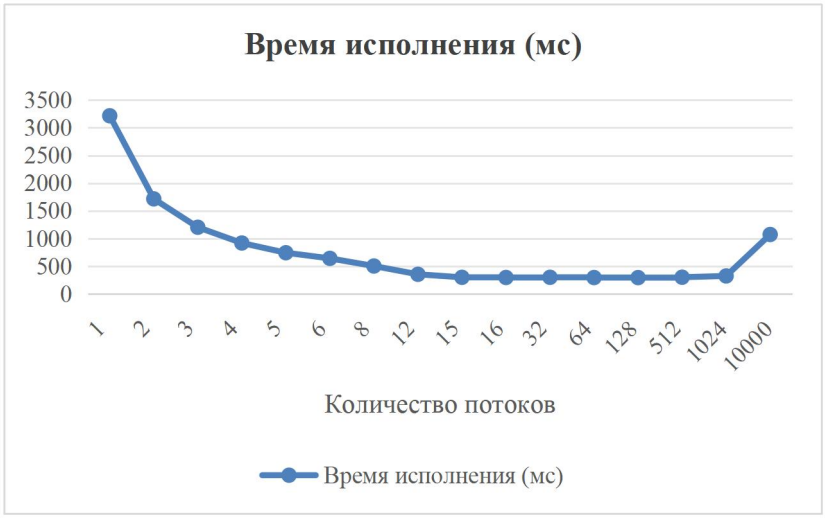

Рисунок 5 – Зависимость времени исполнения от количества потоков

Рисунок 6 – Зависимость ускорения от количества потоков

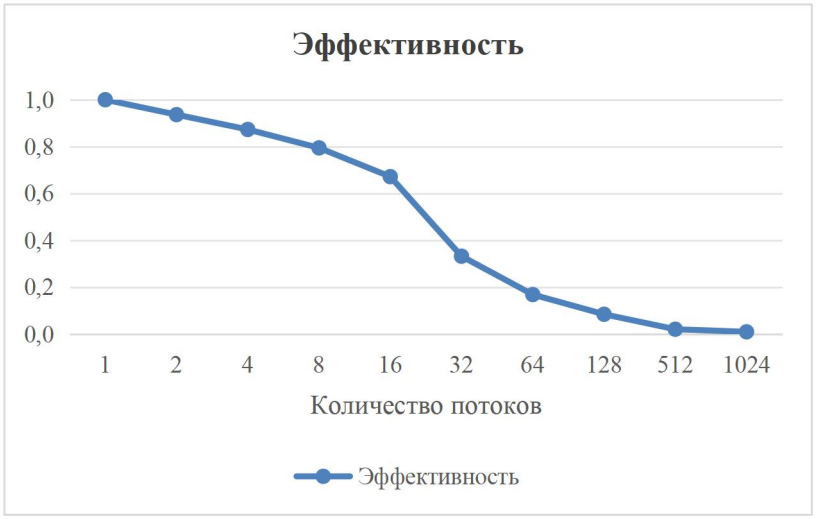

Рисунок 7 – Зависимость эффективности от количества потоков

Из графиков видно, что время исполнения программы заметно снижается, а ускорение растет, при увеличении числа потоков от 1 до количества логических ядер. При дальнейшем увеличении потоков ускорение почти не растет, а затем и вовсе падает. Эффективность уменьшается постоянно, но резко снижается после перехода максимального числа логических ядер.

Код программы

card_model.h:

```
#ifndef CARDS_H
#define CARDS_H

#include <stdbool.h>
#include <stdlib.h>
#include <time.h>

#define CARD_DECK_SIZE 52
#define CARD_PAR1_COUNT 13
#define CARD_PAR2_COUNT 4

typedef struct {
    int parameter_1;
} GameCard_MathModel;

bool random_cards_deck_round(unsigned int* seed_ptr);

bool is_same(const GameCard_MathModel *card1, const GameCard_MathModel
*card2);

#endif
```

card_model.c:

```
#include "../include/card_model.h"

void random_shuffle(GameCard_MathModel *arr, int n, unsigned int*
seed_ptr) {
    for (int i = n - 1; i > 0; --i) {
        int random_ind = rand_r(seed_ptr) % (i + 1);
        GameCard_MathModel tmp = arr[random_ind];
        arr[random_ind] = arr[i];
        arr[i] = tmp;
    }
}

bool is_same(const GameCard_MathModel *card1, const GameCard_MathModel
*card2) {
    return card1->parameter_1 == card2->parameter_1;
}

bool random_cards_deck_round(unsigned int* seed_ptr) {
    GameCard_MathModel cards_deck[CARD_DECK_SIZE];
    int ind = 0;
    for (int i = 0; i < CARD_PAR1_COUNT; ++i) {
        for (int j = 0; j < CARD_PAR2_COUNT; ++j) {
            cards_deck[ind] = (GameCard_MathModel){
                .parameter_1 = i
            };
            ++ind;
```

```
            }
        }
        random_shuffle(cards_deck, CARD_DECK_SIZE, seed_ptr);
        return is_same(&cards_deck[CARD_DECK_SIZE - 1],
&cards_deck[CARD_DECK_SIZE - 2]);
    }
```

time_conv.h:

```
#ifndef TIME_CONV_H
#define TIME_CONV_H

#include <time.h>

double time2ms(
    struct timespec start_time,
    struct timespec end_time
);

#endif
```

time_conv.c:

```
#include "../include/time_conv.h"

double time2ms(
    struct timespec start_time,
    struct timespec end_time
) {
    long long total_ns = (end_time.tv_sec - start_time.tv_sec) *
1000000000LL +
                         (end_time.tv_nsec - start_time.tv_nsec);
    long threads_time_ms = total_ns / 1000000;
    long threads_time_mks = (total_ns % 1000000) / 1000;
    return threads_time_ms + threads_time_mks / (double)1000;
}
```

app.c:

```
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <time.h>

#include "./include/time_conv.h"
#include "./include/card_model.h"

#define TEST_COUNT 3
#define MSG_BUFFERSIZE 256
```

```
    static const char alloc_mem_err_msg[] = "Error: Allocation memory
error.\n";
    static const char create_thread_err_msg[] = "Error: Create thread
error.\n";

    typedef struct {
        int thread_number;
        int rounds_count;
        int satisfactory_case_count;
        unsigned int seed;
        char padding[64 - 3 * sizeof(int) - sizeof(unsigned int)];
    } ThreadArgs;

    int min(int a, int b) {
        if (a < b) {
            return a;
        }
        return b;
    }

    static void *thread_func_do_round(void *_args) {
        ThreadArgs *args = (ThreadArgs*)_args;
        {
            char msg[MSG_BUFFERSIZE];
            size_t msg_size = snprintf(msg, MSG_BUFFERSIZE, "Thread %d
started.\n", args->thread_number);
            write(STDOUT_FILENO, msg, msg_size);
        }
        int satisfactory_case_count = 0;
        for (int completed_rounds = 0; completed_rounds < args->rounds_count;
++completed_rounds) {
            bool cards_sames = random_cards_deck_round(&(args->seed));
            satisfactory_case_count += (int)cards_sames;
        }
        args->satisfactory_case_count = satisfactory_case_count;
        return NULL;
    }

    int parallel_calculation(
        int max_threads_count,
        int rounds_count,
        int *res_threads_count,
        double *res_probability,
        double *res_threads_time_ms
    ) {
        struct timespec start_time, end_time;
        clock_gettime(CLOCK_MONOTONIC, &start_time);

        int threads_count = min(rounds_count, max_threads_count);
        pthread_t *threads = (pthread_t*)malloc(sizeof(pthread_t) *
threads_count);
        ThreadArgs *threads_args = (ThreadArgs*)malloc(sizeof(ThreadArgs) *
threads_count);
        if (threads == NULL || threads_args == NULL) {
```

```
            write(STDERR_FILENO, alloc_mem_err_msg, sizeof(alloc_mem_err_msg)
- 1);
            return -1;
        }
        int min_rounds_count_per_thread = rounds_count / threads_count;
        int extra_rounds_count = rounds_count % threads_count;

        int thread_created = 0;
        for (int i = 0; i < threads_count; ++i) {
            threads_args[i].thread_number = i;
            threads_args[i].rounds_count = min_rounds_count_per_thread +
(int)(i < extra_rounds_count);
            threads_args[i].satisfactory_case_count = 0;
            threads_args[i].seed = (unsigned int)time(NULL) + i * 1000;
            if (pthread_create(&threads[i], NULL, thread_func_do_round,
(void *)&(threads_args[i]))) {
                break;
            }
            ++thread_created;
        }
        for (int i = 0; i < thread_created; ++i) {
            pthread_join(threads[i], NULL);
        }
        if (thread_created < threads_count) {
            write(STDERR_FILENO, create_thread_err_msg,
sizeof(create_thread_err_msg) - 1);
            return -1;
        }

        int satisfactory_case_count = 0;
        for (int i = 0; i < threads_count; ++i) {
            satisfactory_case_count +=
threads_args[i].satisfactory_case_count;
        }
        double probability = satisfactory_case_count / (double)rounds_count;

        free(threads);
        free(threads_args);

        clock_gettime(CLOCK_MONOTONIC, &end_time);
        double threads_time_ms = time2ms(start_time, end_time);

        *res_threads_count = threads_count;
        *res_probability = probability;
        *res_threads_time_ms = threads_time_ms;

        return 0;
    }

    int main(int argc, const char **argv) {
        if (argc - 1 != 2) {
            const char msg[] = "Error: Wrong arguments count. Should be 2:
max_threads_count and rounds_count.\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
```

```
            return -1;
        }
        size_t n_processors = sysconf(_SC_NPROCESSORS_ONLN);
        size_t ThreadArgs_size = sizeof(ThreadArgs);
        {
            char msg[MSG_BUFFERSIZE];
            size_t msg_size = snprintf(msg, MSG_BUFFERSIZE,
                "N_processors: %zu.\n", n_processors);
            write(STDOUT_FILENO, msg, msg_size);
            msg_size = snprintf(msg, MSG_BUFFERSIZE,
                "ThreadArgs size: %zu.\n", ThreadArgs_size);
            write(STDOUT_FILENO, msg, msg_size);
        }
        int max_threads_count = atoi(argv[1]);
        int rounds_count = atoi(argv[2]);

        int threads_count = -1;
        double mean_probability = 0;
        double mean_threads_time_ms = 0;
        for (int i = 0; i < TEST_COUNT; ++i) {
            double probability;
            double threads_time_ms;
            int error = parallel_calculation(
                max_threads_count,
                rounds_count,
                &threads_count,
                &probability,
                &threads_time_ms
            );
            if (error) {
                return error;
            }
            mean_probability += probability / TEST_COUNT;
            mean_threads_time_ms += threads_time_ms / TEST_COUNT;
        }

        {
            int32_t msg_size;
            char msg[MSG_BUFFERSIZE];
            msg_size = snprintf(msg, MSG_BUFFERSIZE,
                "Threads count: %d.\n", threads_count);
            write(STDOUT_FILENO, msg, msg_size);
            msg_size = snprintf(msg, MSG_BUFFERSIZE,
                "Mean probability: %f.\n", mean_probability);
            write(STDOUT_FILENO, msg, msg_size);
            msg_size = snprintf(msg, MSG_BUFFERSIZE,
                "Mean threads execution time: %.3f мс.\n",
mean_threads_time_ms);
            write(STDOUT_FILENO, msg, msg_size);
        }
        return 0;
    }
```

Протокол работы программы

Тестирование

```
root@4bcd85e1faed:/workspaces/OS_labs/lab2/build# ./app 4 10000000

N_processors: 16.
ThreadArgs size: 64.
Thread 0 started.
Thread 1 started.
Thread 2 started.
Thread 3 started.
Thread 0 started.
Thread 1 started.
Thread 2 started.
Thread 3 started.
Thread 0 started.
Thread 1 started.
Thread 2 started.
Thread 3 started.
Threads count: 4.
Mean probability: 0.058910.
Mean threads execution time: 950.315 мс.
```

Strace (4 потока, 10 экспериментов, запуск без подсчета среднего времени исполнения, выделены места создания потоков):

```
root@4bcd85e1faed:/workspaces/OS_labs/lab2/build# strace -
o ../tests/strace_out.txt -f ./app 4 10
```

strace_out.txt:

```
60292 execve("./app", ["./app", "4", "10"], 0x7ffcd16e1db8 /* 27 vars */)
= 0
60292 brk(NULL)                           = 0x9cab000
60292 mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,
-1, 0) = 0x7fdc7bb51000
60292 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or
directory)
60292 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
60292 newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=33091, ...},
AT_EMPTY_PATH) = 0
60292 mmap(NULL, 33091, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fdc7bb48000
60292 close(3)                            = 0
60292 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6",
O_RDONLY|O_CLOEXEC) = 3
60292 read(3,
"\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20t\2\0\0\0\0\0"..., 832) =
832
```

```
    60292 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
    60292 newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=1926232, ...},
AT_EMPTY_PATH) = 0
    60292 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
    60292 mmap(NULL, 1974096, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7fdc7b966000
    60292 mmap(0x7fdc7b98c000, 1400832, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7fdc7b98c000
    60292 mmap(0x7fdc7bae2000, 339968, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17c000) = 0x7fdc7bae2000
    60292 mmap(0x7fdc7bb35000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1cf000) = 0x7fdc7bb35000
    60292 mmap(0x7fdc7bb3b000, 53072, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fdc7bb3b000
    60292 close(3)                          = 0
    60292 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS,
-1, 0) = 0x7fdc7b963000
    60292 arch_prctl(ARCH_SET_FS, 0x7fdc7b963740) = 0
    60292 set_tid_address(0x7fdc7b963a10)   = 60292
    60292 set_robust_list(0x7fdc7b963a20, 24) = 0
    60292 rseq(0x7fdc7b964060, 0x20, 0, 0x53053053) = 0
    60292 mprotect(0x7fdc7bb35000, 16384, PROT_READ) = 0
    60292 mprotect(0x403000, 4096, PROT_READ) = 0
    60292 mprotect(0x7fdc7bb84000, 8192, PROT_READ) = 0
    60292 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0
    60292 munmap(0x7fdc7bb48000, 33091)     = 0
    60292 openat(AT_FDCWD, "/sys/devices/system/cpu/online",
O_RDONLY|O_CLOEXEC) = 3
    60292 read(3, "0-15\n", 1024)           = 5
    60292 close(3)                          = 0
    60292 write(1, "N_processors: 16.\n", 18) = 18
    60292 write(1, "ThreadArgs size: 64.\n", 21) = 21
    60292 getrandom("\xca\xa3\x97\xf8\x74\x47\x9f\xb9", 8, GRND_NONBLOCK) =
8
    60292 brk(NULL)                         = 0x9cab000
    60292 brk(0x9ccc000)                    = 0x9ccc000
    60292 rt_sigaction(SIGRT_1, {sa_handler=0x7fdc7b9ec720, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7fdc7b9a2050}, NULL, 8) = 0
    60292 rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
    60292 mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,
-1, 0) = 0x7fdc7b162000
    60292 mprotect(0x7fdc7b163000, 8388608, PROT_READ|PROT_WRITE) = 0
    60292 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
    60292
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE
_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fdc7b962990, parent_tid=0x7fdc7b962990, exit_signal=0,
```

```
stack=0x7fdc7b162000, stack_size=0x7fff80, tls=0x7fdc7b9626c0}, 88) = -1
ENOSYS (Function not implemented)
    60292 clone(child_stack=0x7fdc7b961f70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM
|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tid=[60293],
tls=0x7fdc7b9626c0, child_tidptr=0x7fdc7b962990) = 60293
    60293 rseq(0x7fdc7b962fe0, 0x20, 0, 0x53053053 <unfinished ...>
    60292 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
    60293 <... rseq resumed>)               = 0
    60292 <... rt_sigprocmask resumed>NULL, 8) = 0
    60293 set_robust_list(0x7fdc7b9629a0, 24 <unfinished ...>
    60292 mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,
-1, 0 <unfinished ...>
    60293 <... set_robust_list resumed>)    = 0
    60292 <... mmap resumed>)               = 0x7fdc7a961000
    60293 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
    60292 mprotect(0x7fdc7a962000, 8388608, PROT_READ|PROT_WRITE
<unfinished ...>
    60293 <... rt_sigprocmask resumed>NULL, 8) = 0
    60292 <... mprotect resumed>)           = 0
    60293 write(1, "Thread 0 started.\n", 18 <unfinished ...>
    60292 rt_sigprocmask(SIG_BLOCK, ~[],  <unfinished ...>
    60293 <... write resumed>)              = 18
    60292 <... rt_sigprocmask resumed>[], 8) = 0
    60293 rt_sigprocmask(SIG_BLOCK, ~[RT_1],  <unfinished ...>
    60292
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE
_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fdc7b161990, parent_tid=0x7fdc7b161990, exit_signal=0,
stack=0x7fdc7a961000, stack_size=0x7fff80, tls=0x7fdc7b1616c0}
<unfinished ...>
    60293 <... rt_sigprocmask resumed>NULL, 8) = 0
    60292 <... clone3 resumed>, 88)         = -1 ENOSYS (Function not
implemented)
    60293 madvise(0x7fdc7b162000, 8368128, MADV_DONTNEED <unfinished ...>
    60292 clone(child_stack=0x7fdc7b160f70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM
|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID <unfinished ...>
    60293 <... madvise resumed>)            = 0
    60292 <... clone resumed>, parent_tid=[60294], tls=0x7fdc7b1616c0,
child_tidptr=0x7fdc7b161990) = 60294
    60294 rseq(0x7fdc7b161fe0, 0x20, 0, 0x53053053 <unfinished ...>
    60293 exit(0 <unfinished ...>
    60292 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
    60294 <... rseq resumed>)               = 0
    60292 <... rt_sigprocmask resumed>NULL, 8) = 0
    60293 <... exit resumed>)               = ?
    60292 mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,
-1, 0 <unfinished ...>
    60294 set_robust_list(0x7fdc7b1619a0, 24 <unfinished ...>
    60292 <... mmap resumed>)               = 0x7fdc7a160000
    60293 +++ exited with 0 +++
    60292 mprotect(0x7fdc7a161000, 8388608, PROT_READ|PROT_WRITE
<unfinished ...>
```

```
    60294 <... set_robust_list resumed>)    = 0
    60292 <... mprotect resumed>)           = 0
    60294 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
    60292 rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
    60294 <... rt_sigprocmask resumed>NULL, 8) = 0
    60292
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE
_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fdc7a960990, parent_tid=0x7fdc7a960990, exit_signal=0,
stack=0x7fdc7a160000, stack_size=0x7fff80, tls=0x7fdc7a9606c0}
<unfinished ...>
    60294 write(1, "Thread 1 started.\n", 18 <unfinished ...>
    60292 <... clone3 resumed>, 88)         = -1 ENOSYS (Function not
implemented)
    60294 <... write resumed>)              = 18
    60292 clone(child_stack=0x7fdc7a95ff70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM
|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID <unfinished ...>
    60294 rt_sigprocmask(SIG_BLOCK, ~[RT_1],  <unfinished ...>
    60292 <... clone resumed>, parent_tid=[60295], tls=0x7fdc7a9606c0,
child_tidptr=0x7fdc7a960990) = 60295
    60294 <... rt_sigprocmask resumed>NULL, 8) = 0
    60292 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
    60295 rseq(0x7fdc7a960fe0, 0x20, 0, 0x53053053 <unfinished ...>
    60294 madvise(0x7fdc7a961000, 8368128, MADV_DONTNEED <unfinished ...>
    60292 <... rt_sigprocmask resumed>NULL, 8) = 0
    60295 <... rseq resumed>)               = 0
    60292 mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK,
-1, 0 <unfinished ...>
    60294 <... madvise resumed>)            = 0
    60292 <... mmap resumed>)               = 0x7fdc7995f000
    60295 set_robust_list(0x7fdc7a9609a0, 24 <unfinished ...>
    60292 mprotect(0x7fdc79960000, 8388608, PROT_READ|PROT_WRITE
<unfinished ...>
    60294 exit(0 <unfinished ...>
    60292 <... mprotect resumed>)           = 0
    60295 <... set_robust_list resumed>)    = 0
    60292 rt_sigprocmask(SIG_BLOCK, ~[],  <unfinished ...>
    60294 <... exit resumed>)               = ?
    60292 <... rt_sigprocmask resumed>[], 8) = 0
    60295 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
    60292
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE
_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fdc7a15f990, parent_tid=0x7fdc7a15f990, exit_signal=0,
stack=0x7fdc7995f000, stack_size=0x7fff80, tls=0x7fdc7a15f6c0}
<unfinished ...>
    60294 +++ exited with 0 +++
    60292 <... clone3 resumed>, 88)         = -1 ENOSYS (Function not
implemented)
    60295 <... rt_sigprocmask resumed>NULL, 8) = 0
    60292 clone(child_stack=0x7fdc7a15ef70,
flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM
|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID <unfinished ...>
```

```
    60295 write(1, "Thread 2 started.\n", 18 <unfinished ...>
    60292 <... clone resumed>, parent_tid=[60296], tls=0x7fdc7a15f6c0,
child_tidptr=0x7fdc7a15f990) = 60296
    60296 rseq(0x7fdc7a15ffe0, 0x20, 0, 0x53053053 <unfinished ...>
    60295 <... write resumed>)              = 18
    60292 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
    60296 <... rseq resumed>)               = 0
    60292 <... rt_sigprocmask resumed>NULL, 8) = 0
    60295 rt_sigprocmask(SIG_BLOCK, ~[RT_1],  <unfinished ...>
    60292 futex(0x7fdc7a960990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME,
60295, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
    60296 set_robust_list(0x7fdc7a15f9a0, 24 <unfinished ...>
    60295 <... rt_sigprocmask resumed>NULL, 8) = 0
    60296 <... set_robust_list resumed>)    = 0
    60295 madvise(0x7fdc7a160000, 8368128, MADV_DONTNEED <unfinished ...>
    60296 rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
    60295 <... madvise resumed>)            = 0
    60296 <... rt_sigprocmask resumed>NULL, 8) = 0
    60295 exit(0 <unfinished ...>
    60296 write(1, "Thread 3 started.\n", 18 <unfinished ...>
    60295 <... exit resumed>)               = ?
    60296 <... write resumed>)              = 18
    60292 <... futex resumed>)              = 0
    60295 +++ exited with 0 +++
    60292 futex(0x7fdc7a15f990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME,
60296, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
    60296 rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
    60296 madvise(0x7fdc7995f000, 8368128, MADV_DONTNEED) = 0
    60296 exit(0)                           = ?
    60292 <... futex resumed>)              = 0
    60296 +++ exited with 0 +++
    60292 write(1, "Threads count: 4.\n", 18) = 18
    60292 write(1, "Mean probability: 0.100000.\n", 28) = 28
    60292 write(1, "Mean threads execution time: 27."..., 42) = 42
    60292 exit_group(0)                     = ?
    60292 +++ exited with 0 +++
```

Вывод

Приобрел практические навыки в управлении потоками в ОС Linux и обеспечении синхронизации между потоками.

Во время выполнения лабораторной работы столкнулся с проблемой увеличения времени исполнения программы на большем количестве потоков из-за синхронизации кеш-линий между ядрами вследствие близкого расположения в памяти структур ThreadArgs, изменяемых разными потоками. Решением стало выравнивание структур до 64 бит, т.е. до размера кеш-линии.

Также было замечено, что в режиме энергосбережения время исполнения программы для любого количества потоков увеличивается приблизительно в два раза.