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-210Б-23

Студент: Домкин П.П.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 14.11.24

Москва, 2024

Постановка задачи

Цель работы

Приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение синхронизации между потоками

Задание

Составить программу на языке Си(C++), 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.

15 вариант:

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

Использованные системные вызовы:

- int pthread_create(pthread_t *__restrict__ __newthread, const pthread_attr_t *__restrict__ __attr, void *(*__start_routine)(void *), void *__restrict__ __arg) — создаёт поток со стартовой функцией и заданными аргументами
- int pthread_join(pthread_t __th, void **__thread_return) — дожидается завершения потока.
- ssize_t write(int __fd, const void *__buf, size_t __n); – Записывает n байт из буфер(buf) в файл (fd). Возвращает количество записанных байт или -1.
- void exit(int __status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.

Для реализации с помощью atomic использовались:

- Библиотека <stdatomic.h>, а именно тип данных atomic_int, макросы atomic_fetch_add(PTR, VAL) и atomic_load(PTR)

Для реализации с помощью mutex использовались:

- pthread_mutex_t – тип данных;

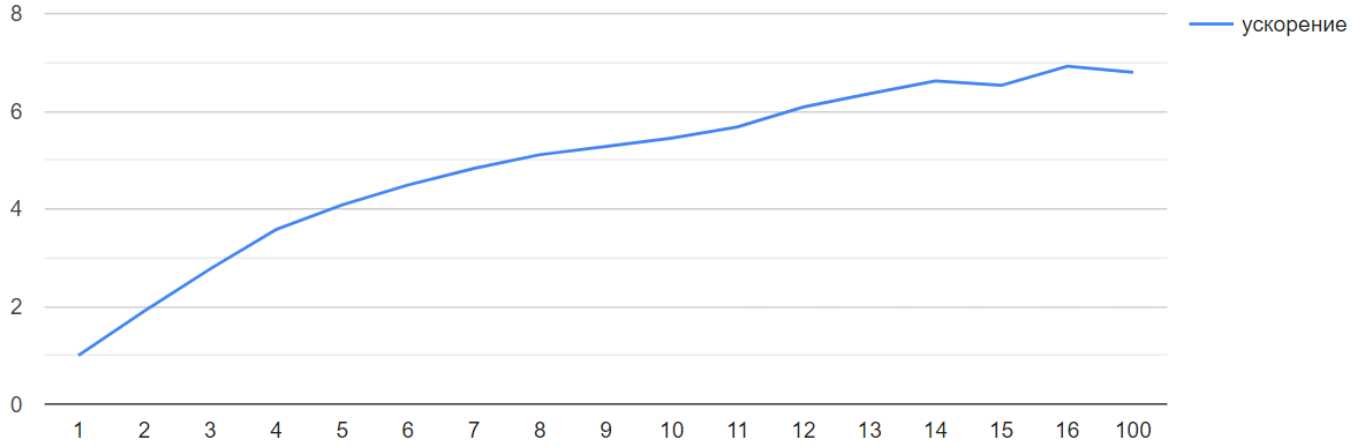
- int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr) – инициализация
- мьютекса;
- int pthread_mutex_lock(pthread_mutex_t *mutex) – блокировка мьютекса;
- int pthread_mutex_unlock(pthread_mutex_t *mutex) – разблокировка мьютекса;
- int pthread_mutex_destroy(pthread_mutex_t *mutex) – удаление мьютекса;

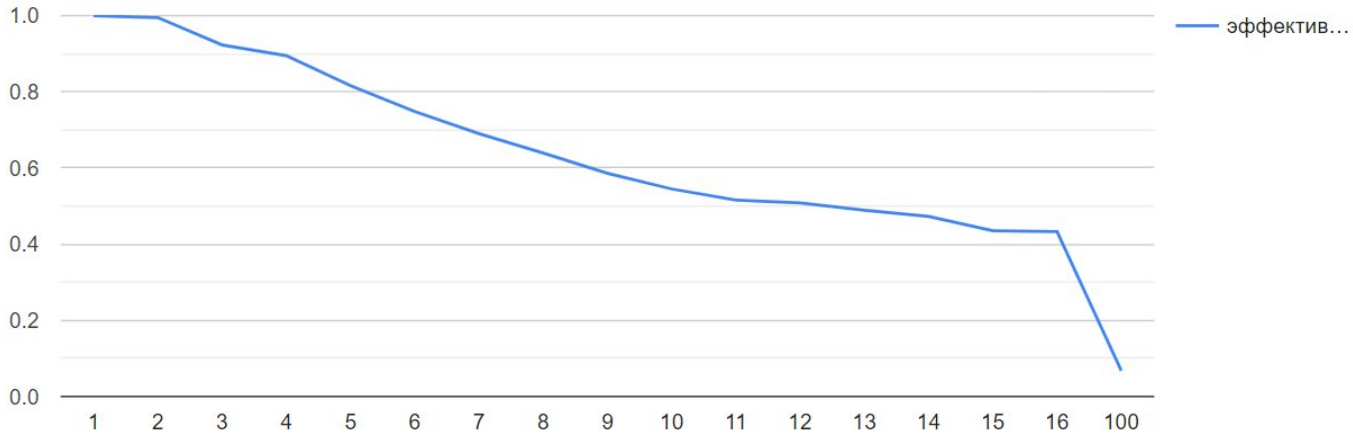
Создаём потоки в количестве указанном в аргументе к программе. Аргументом функции pthread_create будет являться функция ThreadFunction с аргументами Rounds – второй аргумент к программе, seed – семя для рандомайзера для каждого потока. В первой реализации введем атомарный счетчик matches, чтобы не происходил «data race» между потоками, из-за чего кол-во счетчика не совпадает с действительностью (потоки борются за получение доступа к счетчику). Во второй реализации используем mutex, который ограничивает доступ потоков к обычной переменной matches и позволяет только одному потоку использовать данную переменную в один момент.

Метод Монте-Карло заключается в оперировании случайностями. По заданию необходимо найти вероятность того, что первые две карты в колоде из 52 карт будут одинаковой масти. Это можно рассчитать, проведя эксперимент — взять колоду, перемешать её и проверить что верхние карты одинаковы. Колода перемешивается с помощью радномайзера. В удачном случае счетчик matches будем увеличивать атомарно или с mutex. Вероятность будем рассчитывать как результат деления количества удачный исходов на общее количество раундов.

Rounds = 100000000

число потоков	время выполнения, мс	ускорение	Эффективность1
1	17690	1,00	1,00
2	9214	1,91	0,995
3	6378	2,77	0,923
4	4939	3,58	0,895
5	4332	4,08	0,816
6	3940	4,49	0,748
7	3661	4,83	0,69
8	3459	5,11	0,639
9	3349	5,28	0,586
10	3223	5,45	0,545
11	3114	5,68	0,516
12	2902	6,09	0,508
13	2779	6,36	0,489
14	2671	6,62	0,473
15	2710	6,53	0,435
16	2556	6,92	0,433
100	2600	6,80	0,068


8
6
4
2
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
100
ускорение


1.0
0.8
0.6
0.4
0.2
0.0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
100
эффектив…

Код программы

main.c (atomic)

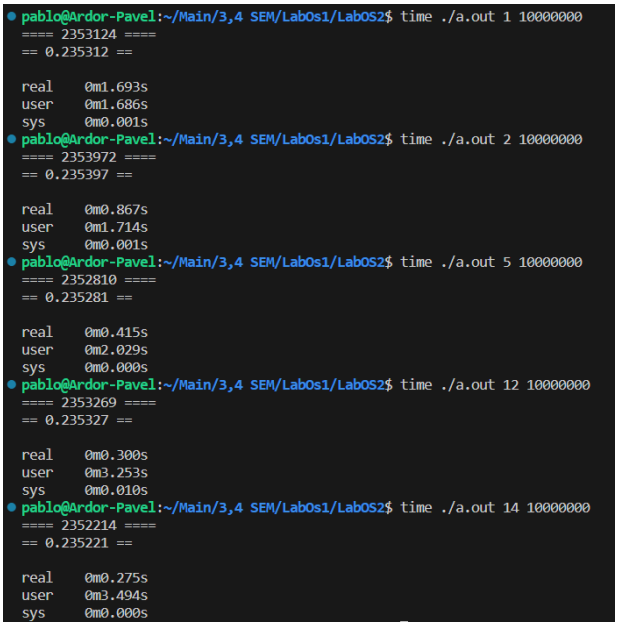
```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <unistd.h>
#include <stdarg.h>
#include <time.h>
#include <stdatomic.h>

#define DECK_NUM 52

atomic_int matches = 0;

void PrintHelper(const char *fmt, ...) {
    va_list args;
    va_start(args, fmt);
    char buffer[BUFSIZ];
    vsnprintf(buffer, sizeof(buffer), fmt, args);
    if (write(STDOUT_FILENO, buffer, strlen(buffer)) == -1) {
        exit(EXIT_FAILURE);
    }
    va_end(args);
}

void *ThreadFunction(void *args) {
    size_t round = ((size_t *)args)[0];
    unsigned int seed = ((size_t *)args)[1];
    int local = 0;
    int deck[DECK_NUM];
    for (int j = 0; j < DECK_NUM; j++) {
        deck[j] = j;
    }

    for (size_t i = 0; i < round; i++) {
        for (int j = DECK_NUM - 1; j > 0; j--) {
            int k = rand_r(&seed) % (j + 1);
            int temp = deck[j];
            deck[j] = deck[k];
            deck[k] = temp;
        }

        if (deck[0] % 4 == deck[1] % 4) {
            ++local;
        }
    }

    atomic_fetch_add(&matches, local);
    return NULL;
}

int main(int argc, char **argv) {
    if (argc != 3) {
        PrintHelper("Input error. Use: %s <threads> <rounds>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    size_t threads_num = atol(argv[1]);
    size_t rounds = atol(argv[2]);
    size_t rounds_for_thread = rounds / threads_num;
    pthread_t threads[threads_num];
    size_t thread_args[threads_num][2];

    // Creating threads
    for (size_t i = 0; i < threads_num; i++) {
        thread_args[i][0] = rounds_for_thread;
        thread_args[i][1] = time(NULL) ^ (i + 1);
        if (pthread_create(&threads[i], NULL, ThreadFunction, thread_args[i]) != 0) {
            PrintHelper("Error. Thread %zu not created\n", i);
            exit(EXIT_FAILURE);
        }
    }

    // Waiting for finishing threads
    for (size_t i = 0; i < threads_num; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            PrintHelper("Error. Thread not joined\n");
            exit(EXIT_FAILURE);
        }
    }

    atomic_int count_matches = atomic_load(&matches);
    PrintHelper("==== %d ====\n", count_matches);

    double probability = (double)count_matches / (double)rounds;
    PrintHelper("== %.6f ==\n", probability);

    return EXIT_SUCCESS;
}
```

main_copy.c (mutex)

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <unistd.h>
#include <stdarg.h>
#include <time.h>

#define DECK_NUM 52

int matches = 0;
pthread_mutex_t m;

void PrintHelper(const char *fmt, ...) {
    va_list args;
    va_start(args, fmt);
    char buffer[BUFSIZ];
    vsnprintf(buffer, sizeof(buffer), fmt, args);
    if (write(STDOUT_FILENO, buffer, strlen(buffer)) == -1) {
        exit(EXIT_FAILURE);
    }
    va_end(args);
}

void *ThreadFunction(void *args) {
    size_t round = *(size_t *)args;
    unsigned int seed = ((size_t *)args)[1];
    int local = 0;
    int deck[DECK_NUM];
    for (int j = 0; j < DECK_NUM; j++) {
        deck[j] = j;
    }

    for (size_t i = 0; i < round; i++) {
        for (int j = DECK_NUM - 1; j > 0; j--) {
            int k = rand_r(&seed) % (j + 1);
            int temp = deck[j];
            deck[j] = deck[k];
            deck[k] = temp;
        }

        if (deck[0] % 4 == deck[1] % 4) {
            ++local;
        }
    }

    if (pthread_mutex_lock(&m) != 0){
        PrintHelper("Error lock mutex.\n");
        exit(EXIT_FAILURE);
    }

    matches += local;

    if (pthread_mutex_unlock(&m) != 0){
        PrintHelper("Error unlock mutex.\n");
        exit(EXIT_FAILURE);
    }

    return NULL;
}

int main(int argc, char **argv) {
    if (argc != 3) {
        PrintHelper("Input error. Use: %s <threads> <rounds>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    size_t threads_num = atol(argv[1]);
    size_t rounds = atol(argv[2]);
    size_t rounds_for_thread = rounds / threads_num;
    pthread_t threads[threads_num];

    if (pthread_mutex_init(&m, NULL)) {
        PrintHelper("Error create mutex.\n");
        exit(EXIT_FAILURE);
    }

    for (size_t i = 0; i < threads_num; i++) {
        if (pthread_create(&threads[i], NULL, ThreadFunction, &rounds_for_thread) != 0) {
            PrintHelper("Error. Thread %zu not created\n", i);
            exit(EXIT_FAILURE);
        }
    }

    for (size_t i = 0; i < threads_num; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            PrintHelper("Error. Thread not joined\n");
            exit(EXIT_FAILURE);
        }
    }

    pthread_mutex_destroy(&m);

    PrintHelper("==== %d ====\n", matches);

    double probability = (double)matches / (double)rounds;
    PrintHelper("== %.6f ==\n", probability);

    return EXIT_SUCCESS;
}
```

Протокол работы программы

Тестирование:



Strace:

pablo@Ardor-Pavel:~/Main/3,4 SEM/LabOs1/LabOS2$ strace ./a.out 5 1000000

execve("./a.out", ["./a.out", "5", "1000000"], 0x7ffee83e7f80 /* 35 vars */) = 0

brk(NULL) = 0x557ea9ea7000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdf8aed290) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f7d540b2000

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=18823, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 18823, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f7d540ad000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

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

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

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

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f7d53e84000

mprotect(0x7f7d53eac000, 2023424, PROT_NONE) = 0

mmap(0x7f7d53eac000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f7d53eac000

mmap(0x7f7d54041000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f7d54041000

mmap(0x7f7d5409a000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f7d5409a000

mmap(0x7f7d540a0000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f7d540a0000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f7d53e81000

arch_prctl(ARCH_SET_FS, 0x7f7d53e81740) = 0

set_tid_address(0x7f7d53e81a10) = 29198

set_robust_list(0x7f7d53e81a20, 24) = 0

rseq(0x7f7d53e820e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f7d5409a000, 16384, PROT_READ) = 0

mprotect(0x557e6e945000, 4096, PROT_READ) = 0

mprotect(0x7f7d540ec000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7f7d540ad000, 18823) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x7f7d53f15870, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f7d53ec6520}, NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f7d53680000

mprotect(0x7f7d53681000, 8388608, PROT_READ|PROT_WRITE) = 0

getrandom("\xeb\xed\xdd\x82\x14\x99\x8b\xd6", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x557ea9ea7000

brk(0x557ea9ec8000) = 0x557ea9ec8000

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLE

ARTID, child_tid=0x7f7d53e80910, parent_tid=0x7f7d53e80910, exit_signal=0, stack=0x7f7d53680000, stack_size=0x7fff00, tls=0x7f7d53e80640} => {parent_tid=[29199]}, 88) = 29199

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f7d52e7f000

mprotect(0x7f7d52e80000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f7d5367f910, parent_tid=0x7f7d5367f910, exit_signal=0, stack=0x7f7d52e7f000, stack_size=0x7fff00, tls=0x7f7d5367f640} => {parent_tid=[29200]}, 88) = 29200

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f7d5267e000

mprotect(0x7f7d5267f000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f7d52e7e910, parent_tid=0x7f7d52e7e910, exit_signal=0, stack=0x7f7d5267e000, stack_size=0x7fff00, tls=0x7f7d52e7e640} => {parent_tid=[29201]}, 88) = 29201

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f7d51e7d000

mprotect(0x7f7d51e7e000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f7d5267d910, parent_tid=0x7f7d5267d910, exit_signal=0, stack=0x7f7d51e7d000, stack_size=0x7fff00, tls=0x7f7d5267d640} => {parent_tid=[29202]}, 88) = 29202

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f7d5167c000

mprotect(0x7f7d5167d000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f7d51e7c910, parent_tid=0x7f7d51e7c910, exit_signal=0,

stack=0x7f7d5167c000, stack_size=0x7fff00, tls=0x7f7d51e7c640} => {parent_tid=[29203]}, 88) = 29203

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

futex(0x7f7d53e80910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 29199, NULL, FUTEX_BITSET_MATCH_ANY) = 0

futex(0x7f7d5367f910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 29200, NULL, FUTEX_BITSET_MATCH_ANY) = 0

futex(0x7f7d52e7e910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 29201, NULL, FUTEX_BITSET_MATCH_ANY) = 0

futex(0x7f7d5267d910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 29202, NULL, FUTEX_BITSET_MATCH_ANY) = 0

munmap(0x7f7d53680000, 8392704) = 0

write(1, "==== 235240 ====\n", 17==== 235240 ====

) = 17

write(1, "== 0.235240 ==\n", 15== 0.235240 ==

) = 15

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе написания данной лабораторной работы я научился создавать программы, работающие с несколькими потоками, а также научился синхронизировать их между собой. В ходе тестирования программы я проанализировал влияние количества потоков на её производительность и ускорение. Выяснилось, что при большом объёме входных данных значительное количество потоков обеспечивает хорошее ускорение, однако эффективное использование ресурсов наблюдается только при небольшом числе потоков, не превышающем количества логических ядер процессора. Выполнение лабораторной работы оказалось интересным опытом, так как я впервые работал с многопоточностью и синхронизацией в языке Си.