

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-211Б-23

Студент: Рожков И.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 10.11.24

Москва, 2024

Постановка задачи

Цель работы:

Целью является приобретение практических навыков в:

- Управление потоками в ОС
- Обеспечение 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

Общий метод и алгоритм решения

Использованные системные вызовы:

- `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.
- `int pthread_create(pthread_t *__restrict, pthread_attr_t *__restrict, void *(*__start_routine)(void *), void *__restrict __arg)` — создаёт поток с рутиной (стартовой функцией) и заданными аргументами
- `int pthread_join(pthread_t __th, void **__thread_return)` — дожидается завершения потока.

Также для атомик реализации были использованы тип данных `atomic_int` и макрос `atomic_fetch_add(PTR,VAL)` из стандартной библиотеки `<stdatomic.h>`. Для mutex реализации были использованы:

`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;

Программа получает на вход два аргумента – максимальное количество потоков и количество раундов. Для полученных значений поровну распределяется количество раундов на каждый поток. Далее создаётся и заполняется массив для хранения всей колоды из 52 карт.

После создаётся нужное количество потоков, которые выполняют функцию `check_probability`. Функция для каждого раунда определяет два псевдо случайных индекса, имитируя взятия двух первых карт с перемешанной колоды. После суммирования успешных событий, результат прибавляется в глобальную переменную `success_events`. Для синхронизации потоков используется атомарный тип `int` в одной реализации и `mutex` в другой.

После проверки завершения всех процессов с помощью функции `pthread_join`, программа выводит результат на экран.

Суть самого метода Монте-Карло заключается в следующем: процесс моделируется с использованием генератора случайных величин, модель многократно обчисляется, на основе полученных данных вычисляются вероятностные характеристики рассматриваемого процесса.

Данные значения приведены для обеих реализаций вместе, так как разница между их тестами не превышает погрешности между двумя запусками одной программы с одинаковыми входными данными.

Число потоков	Время выполнения	Ускорение	Эффективность
1	37781	1,00	1,00
2	12752	2,96	1,48
3	8662	4,36	1,45
4	7077	5,34	1,33
5	6727	5,62	1,12
6	5822	6,49	1,08
7	5499	6,87	0,98
8	5155	7,33	0,92
20	4363	8,66	0,43

Количество раундов	Время выполнения(мс)
100	73
1000	81
10000	92
100000	107
1000000	168
10000000	472
100000000	4036
1000000000	44021



Код программы

atomic:

```
#include "stdio.h"
#include <stdlib.h>
#include <pthread.h>
#include <stdatomic.h>
#include <unistd.h>
#include <string.h>

atomic_int success_events = 0;

typedef struct Cards
{
    int suit;
    int ranks;
} Cards;

typedef struct arg_struct
{
    Cards *cards_arr;
    int count_tests;
} arg_t;

void print(const char *text)
{
    if (!text)
        return;

    if (write(STDOUT_FILENO, text, strlen(text)) == -1)
        exit(EXIT_FAILURE);
}
```

```

}

void *check_probability(void *__args)
{
    arg_t *args = (arg_t *)__args;
    int count_succes = 0;
    int idx_1, idx_2;
    srand((unsigned)time(NULL));
    for (int i = 0; i < args->count_tests; ++i)
    {
        idx_1 = rand() % 52;
        do
        {
            idx_2 = rand() % 52;
        } while (idx_2 == idx_1);
        if (args->cards_arr[idx_1].suit == args->cards_arr[idx_2].suit)
            count_succes++;
    }
    atomic_fetch_add(&success_events, count_succes);
    return NULL;
}

void create_cards_arr(Cards *cards_arr)
{
    int idx = 0;
    if (!cards_arr)
        return;

    for (int i = 1; i < 14; ++i)
    {
        for (int j = 0; j < 4; ++j)
        {
            cards_arr[idx].ranks = i;
            cards_arr[idx++].suit = j;
        }
    }
}

```

```

    }

}

}

int main(int argc, char **argv)
{
    int max_count_treads, count_rounds, remainder;

    if (argc != 3)
    {
        print("Input error. Enter
<program_name><max_count_treads><count_rounds>\n");
        exit(EXIT_FAILURE);
    }

    max_count_treads = atoi(argv[1]);
    count_rounds = atoi(argv[2]);

    pthread_t treads[max_count_treads];
    Cards cards_arr[52];
    create_cards_arr(cards_arr);

    remainder = count_rounds % max_count_treads;

    arg_t args = {.cards_arr = cards_arr, .count_tests = count_rounds /
max_count_treads};

    if (remainder)
    {
        args.count_tests += remainder;

        if (pthread_create(&treads[0], NULL, check_probability, (void *)(&args)))
        {
            print("Pthread_create error\n");
            exit(EXIT_FAILURE);
        }

        args.count_tests -= remainder;
    }
}

```

```
for (int i = (remainder) ? 1 : 0; i < max_count_treads; ++i)
{
    if (pthread_create(&treads[i], NULL, check_probability, (void *)(&args)))
    {
        print("Pthread_create error\n");
        exit(EXIT_FAILURE);
    }
}

for (int i = 0; i < max_count_treads; ++i)
{
    if (pthread_join(treads[i], NULL))
    {
        print("Pthread_join error\n");
        exit(EXIT_FAILURE);
    }
}

char result[100];
sprintf(result, "%.3lf%%\n", (double)success_events / count_rounds * 100);
print(result);
return 0;
}
```


mutex:

```
#include "stdio.h"

#include <stdlib.h>

#include <pthread.h>

#include <stdatomic.h>

#include <unistd.h>

#include <string.h>


int success_events = 0;

pthread_mutex_t m;


typedef struct Cards
{
    int suit;
    int ranks;
} Cards;


typedef struct arg_struct
{
    Cards *cards_arr;
    int count_tests;
} arg_t;


void print(const char *text)
{
    if (!text)
        return;

    if (write(STDOUT_FILENO, text, strlen(text)) == -1)
```

```

        exit(EXIT_FAILURE);
    }

void *check_probability(void *__args)
{
    arg_t *args = (arg_t *)__args;

    int count_succes = 0;

    int idx_1, idx_2;

    srand((unsigned)time(NULL));

    for (int i = 0; i < args->count_tests; ++i)
    {
        idx_1 = rand() % 52;

        do
        {
            idx_2 = rand() % 52;

        } while (idx_2 == idx_1);

        if (args->cards_arr[idx_1].suit == args->cards_arr[idx_2].suit)
            count_succes++;
    }

    if (pthread_mutex_lock(&m))
    {
        print("Pthread_mutex_lock error\n");

        exit(EXIT_FAILURE);
    }

    success_events += count_succes;

    if (pthread_mutex_unlock(&m))
    {
        print("Pthread_mutex_unlock error\n");

        exit(EXIT_FAILURE);
    }
}

```

```

    }

    return NULL;
}

void create_cards_arr(Cards *cards_arr)
{
    int idx = 0;

    if (!cards_arr)
        return;

    for (int i = 1; i < 14; ++i)
    {
        for (int j = 0; j < 4; ++j)
        {
            cards_arr[idx].ranks = i;
            cards_arr[idx++].suit = j;
        }
    }
}

int main(int argc, char **argv)
{
    int max_count_treads, count_rounds, remainder;

    if (argc != 3)
    {
        print("Input error. Enter <program_name><max_count_treads><count_rounds>\n");
        exit(EXIT_FAILURE);
    }

    max_count_treads = atoi(argv[1]);

```

```
count_rounds = atoi(argv[2]);

if (pthread_mutex_init(&m, NULL))

{

    print("Pthread_mutex_create error\n");

    exit(EXIT_FAILURE);

}


pthread_t treads[max_count_treads];

Cards cards_arr[52];

create_cards_arr(cards_arr);


remainder = count_rounds % max_count_treads;

arg_t args = {.cards_arr = cards_arr, .count_tests = count_rounds /
max_count_treads};


if (remainder)

{

    args.count_tests += remainder;

    if (pthread_create(&treads[0], NULL, check_probability, (void *)(&args)))

    {

        print("Pthread_create error\n");

        exit(EXIT_FAILURE);

    }

    args.count_tests -= remainder;

}


for (int i = (remainder) ? 1 : 0; i < max_count_treads; ++i)

{

    if (pthread_create(&treads[i], NULL, check_probability, (void *)(&args)))
```

```

    {

        print("Pthread_create error\n");

        exit(EXIT_FAILURE);

    }

}

for (int i = 0; i < max_count_treads; ++i)
{
    if (pthread_join(treads[i], NULL))
    {
        print("Pthread_join error\n");

        exit(EXIT_FAILURE);

    }
}

char result[100];

if (pthread_mutex_destroy(&m))
{
    print("Pthread_mutex_destroy error\n");

    exit(EXIT_FAILURE);

}

sprintf(result, "%.3lf%%\n", (double)success_events / count_rounds * 100);

print(result);

return 0;

}

```

Протокол работы программы

```
empress@empress:~/OS_labs/lab_2/src$ time ./main.exe 1 34322
23.559%
```

```
real    0m0.099s
user    0m0.000s
sys     0m0.003s
```

```
empress@empress:~/OS_labs/lab_2/src$ time ./main.exe 1 44444
23.488%
```

```
real    0m0.091s
user    0m0.000s
sys     0m0.003s
```

```
empress@empress:~/OS_labs/lab_2/src$ time ./main.exe 1 2323323
23.538%
```

```
real    0m0.240s
user    0m0.003s
sys     0m0.000s
```

```
empress@empress:~/OS_labs/lab_2/src$ time ./main.exe 10 999999999
23.532%
```

```
real    0m9.682s
user    0m0.003s
sys     0m0.000s
```

```
empress@empress:~/OS_labs/lab_2/src$ time ./main.exe 10 1000000
23.391%
```

```
real    0m0.131s
user    0m0.003s
sys     0m0.000s
```

```
empress@empress:~/OS_labs/lab_2/src$ time ./main.exe 1 1000000
23.509%
```

```
real    0m0.129s
user    0m0.003s
sys     0m0.000s
```

```
empress@empress:~/OS_labs/lab_2/src$ time ./main.exe 8 19999999
23.528%
```

```
real    0m0.255s
user    0m0.003s
sys     0m0.000s
```

```
empress@empress:~/OS_labs/lab_2/src$ time ./main.exe 1 19999999
23.525%
```

```
real    0m1.146s
```



```

mmap(0x7f8d6cdbb000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7f8d6cdbb000

mmap(0x7f8d6cdc1000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7f8d6cdc1000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f8d6cba2000

arch_prctl(ARCH_SET_FS, 0x7f8d6cba2740) = 0

set_tid_address(0x7f8d6cba2a10) = 295471

set_robust_list(0x7f8d6cba2a20, 24) = 0

rseq(0x7f8d6cba30e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f8d6cdbb000, 16384, PROT_READ) = 0

mprotect(0x55fcd088b000, 4096, PROT_READ) = 0

mprotect(0x7f8d6ce0d000, 8192, PROT_READ) = 0

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

munmap(0x7f8d6cdce000, 18103) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x7f8d6cc36870, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f8d6cbe7520}, 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) =
0x7f8d6c3a1000

mprotect(0x7f8d6c3a2000, 8388608, PROT_READ|PROT_WRITE) = 0

getrandom("\x23\x40\xcb\x53\xb1\xcf\x37\x74", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x55fcd131a000

brk(0x55fcd133b000) = 0x55fcd133b000

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

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d6cba1910,
parent_tid=0x7f8d6cba1910, exit_signal=0, stack=0x7f8d6c3a1000, stack_size=0x7fff00,
tls=0x7f8d6cba1640} => {parent_tid=[295472]}, 8) = 295472

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d6bba0000

mprotect(0x7f8d6bba1000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d6c3a0910,
parent_tid=0x7f8d6c3a0910, exit_signal=0, stack=0x7f8d6bba0000, stack_size=0x7fff00,
tls=0x7f8d6c3a0640} => {parent_tid=[295473]}, 8) = 295473

```



```

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d6b39f000

mprotect(0x7f8d6b3a0000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d6bb9f910,
parent_tid=0x7f8d6bb9f910, exit_signal=0, stack=0x7f8d6b39f000, stack_size=0x7fff00,
tls=0x7f8d6bb9f640} => {parent_tid=[295474]}, 88) = 295474

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d6ab9e000

mprotect(0x7f8d6ab9f000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d6b39e910,
parent_tid=0x7f8d6b39e910, exit_signal=0, stack=0x7f8d6ab9e000, stack_size=0x7fff00,
tls=0x7f8d6b39e640} => {parent_tid=[295475]}, 88) = 295475

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d6a39d000

mprotect(0x7f8d6a39e000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d6ab9d910,
parent_tid=0x7f8d6ab9d910, exit_signal=0, stack=0x7f8d6a39d000, stack_size=0x7fff00,
tls=0x7f8d6ab9d640} => {parent_tid=[295476]}, 88) = 295476

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d69b9c000

mprotect(0x7f8d69bd000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d6a39c910,
parent_tid=0x7f8d6a39c910, exit_signal=0, stack=0x7f8d69b9c000, stack_size=0x7fff00,
tls=0x7f8d6a39c640} => {parent_tid=[295477]}, 88) = 295477

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d6939b000

mprotect(0x7f8d6939c000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d69b9b910,
parent_tid=0x7f8d69b9b910, exit_signal=0, stack=0x7f8d6939b000, stack_size=0x7fff00,
tls=0x7f8d69b9b640} => {parent_tid=[295478]}, 88) = 295478

```

```

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d68b9a000

mprotect(0x7f8d68b9b000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d6939a910,
parent_tid=0x7f8d6939a910, exit_signal=0, stack=0x7f8d68b9a000, stack_size=0x7fff00,
tls=0x7f8d6939a640} => {parent_tid=[295479]}, 88) = 295479

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d68399000

mprotect(0x7f8d6839a000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d68b99910,
parent_tid=0x7f8d68b99910, exit_signal=0, stack=0x7f8d68399000, stack_size=0x7fff00,
tls=0x7f8d68b99640} => {parent_tid=[295480]}, 88) = 295480

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d67b98000

mprotect(0x7f8d67b99000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d68398910,
parent_tid=0x7f8d68398910, exit_signal=0, stack=0x7f8d67b98000, stack_size=0x7fff00,
tls=0x7f8d68398640} => {parent_tid=[295481]}, 88) = 295481

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d67397000

mprotect(0x7f8d67398000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d67b97910,
parent_tid=0x7f8d67b97910, exit_signal=0, stack=0x7f8d67397000, stack_size=0x7fff00,
tls=0x7f8d67b97640} => {parent_tid=[295482]}, 88) = 295482

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d66b96000

mprotect(0x7f8d66b97000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d67396910,
parent_tid=0x7f8d67396910, exit_signal=0, stack=0x7f8d66b96000, stack_size=0x7fff00,
tls=0x7f8d67396640} => {parent_tid=[295483]}, 88) = 295483

```

```

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d66395000

mprotect(0x7f8d66396000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d66b95910,
parent_tid=0x7f8d66b95910, exit_signal=0, stack=0x7f8d66395000, stack_size=0x7fff00,
tls=0x7f8d66b95640} => {parent_tid=[295484]}, 88) = 295484

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d65b94000

mprotect(0x7f8d65b95000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d66394910,
parent_tid=0x7f8d66394910, exit_signal=0, stack=0x7f8d65b94000, stack_size=0x7fff00,
tls=0x7f8d66394640} => {parent_tid=[295485]}, 88) = 295485

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

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7f8d65393000

mprotect(0x7f8d65394000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f8d65b93910,
parent_tid=0x7f8d65b93910, exit_signal=0, stack=0x7f8d65393000, stack_size=0x7fff00,
tls=0x7f8d65b93640} => {parent_tid=[295486]}, 88) = 295486

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

futex(0x7f8d6cba1910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 295472, NULL,
FUTEX_BITSET_MATCH_ANY) = 0

futex(0x7f8d6c3a0910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 295473, NULL,
FUTEX_BITSET_MATCH_ANY) = 0

futex(0x7f8d6ab9d910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 295476, NULL,
FUTEX_BITSET_MATCH_ANY) = 0

munmap(0x7f8d6c3a1000, 8392704) = 0

futex(0x7f8d6a39c910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 295477, NULL,
FUTEX_BITSET_MATCH_ANY) = 0

munmap(0x7f8d6bba0000, 8392704) = 0

munmap(0x7f8d6b39f000, 8392704) = 0

munmap(0x7f8d6ab9e000, 8392704) = 0

munmap(0x7f8d6a39d000, 8392704) = 0

munmap(0x7f8d69b9c000, 8392704) = 0

munmap(0x7f8d6939b000, 8392704) = 0

```

```
munmap(0x7f8d68b9a000, 8392704)      = 0
munmap(0x7f8d68399000, 8392704)      = 0
munmap(0x7f8d67b98000, 8392704)      = 0
munmap(0x7f8d67397000, 8392704)      = 0
write(1, "23.51\n", 90.235099
)                                     = 9
exit_group(0)                       = ?
+++ exited with 0 +++
```

Вывод

В ходе написания данной лабораторной работы я научился создавать программы, работающие с несколькими потоками, а также синхронизировать их между собой. В результате тестирования программы, я проанализировал каким образом количество потоков влияет на эффективность и ускорение работы программы. Оказалось, что большое количество потоков даёт хорошее ускорение на больших количествах входных данных, но эффективность использования ресурсов находится на приемлемом уровне только на небольшом количестве потоков, не превышающем количества логических ядер процессора. Лабораторная работа была довольно интересна, так как я впервые работал с многопоточностью и синхронизацией на СИ.