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 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 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\_\_ \_\_newthread, const pthread\_attr\_t \*\_\_restrict\_\_ \_\_attr, 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 рандомных 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.

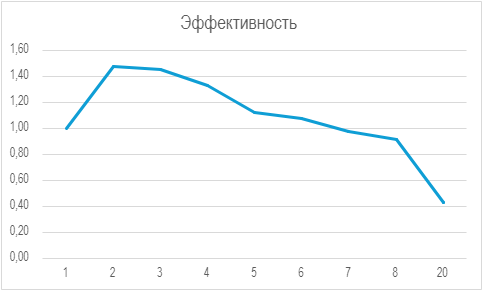
После проверки завершения всех процессов с помощью функции pthread\_join, программа выводит результат на экран.

Суть самого метода Монте-Карло заключается в следующем: процесс моделируется с использованием генератора случайных величин, модель многократно обсчитывается, 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 9999999999

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

user 0m0.003s

sys 0m0.000s

empress@empress:~/OS\_labs/lab\_2/src$ time ./main.exe 1000 2140000000

23.517%

real 0m8.888s

user 0m0.003s

sys 0m0.000s

empress@empress:~/OS\_labs/lab\_2/src$ time ./main.exe 10 2140000000

23.531%

real 0m9.771s

user 0m0.003s

sys 0m0.000s

execve("./main", ["./main", "15", "1000000"], 0x7ffd22586270 /\* 35 vars \*/) = 0

brk(NULL) = 0x55fcd131a000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fff1db58b40) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f8d6cdd3000

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=18103, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 18103, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f8d6cdce000

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) = 0x7f8d6cba5000

mprotect(0x7f8d6cbcd000, 2023424, PROT\_NONE) = 0

mmap(0x7f8d6cbcd000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f8d6cbcd000

mmap(0x7f8d6cd62000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f8d6cd62000

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|CLONE\_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]}, 88) = 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|CLONE\_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]}, 88) = 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|CLONE\_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|CLONE\_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|CLONE\_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(0x7f8d69b9d000, 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=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|CLONE\_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|CLONE\_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|CLONE\_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|CLONE\_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|CLONE\_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|CLONE\_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|CLONE\_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|CLONE\_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|CLONE\_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.