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(С++), 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:

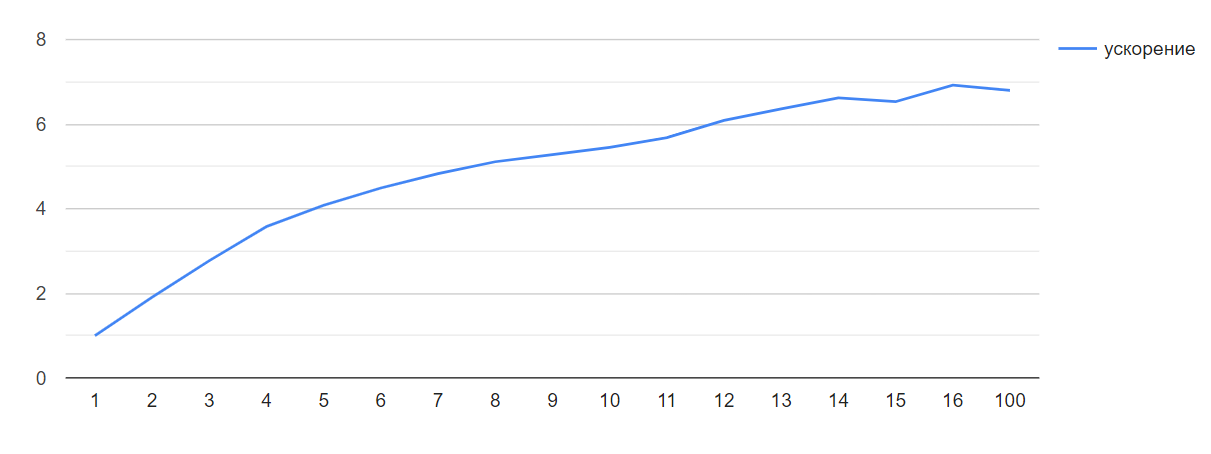
* pthread\_mutex\_t – тип данных;
* int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t \*mutexattr) – инициализация
* мьютекса;
* int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex) – блокировка мьютекса;
* int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex) – разблокировка мьютекса;
* int pthread\_mutex\_destroy(pthread\_mutex\_t \*mutex) – удаление мьютекса;

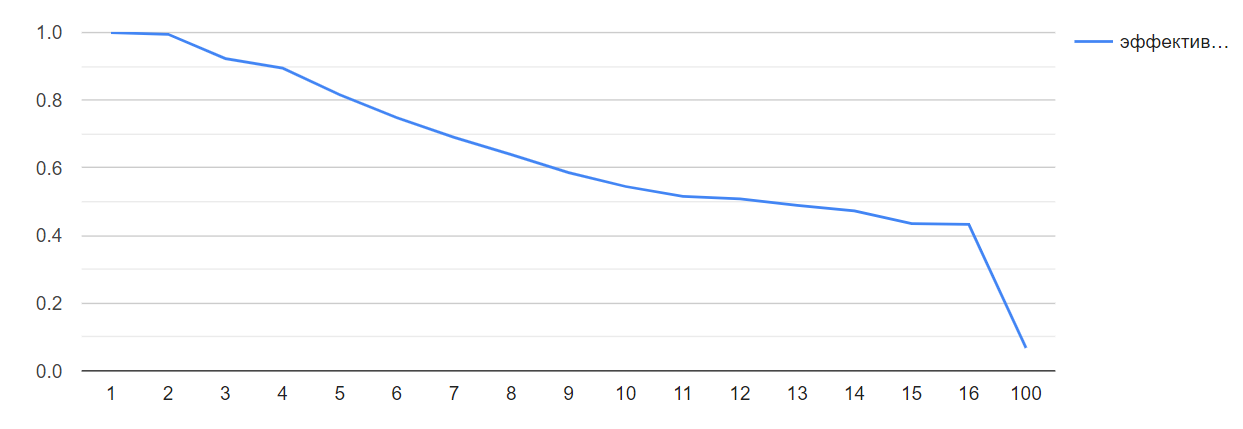
Создаём потоки в количестве указанном в аргументе к программе. Аргументом функции pthread\_create будет являться функция ThreadFunction с аргументами Rounds – второй аргумент к программе, seed – семя для рандомайзера для каждого потока. В первой реализации введем атомарный счетчик matches, чтобы не происходил «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.

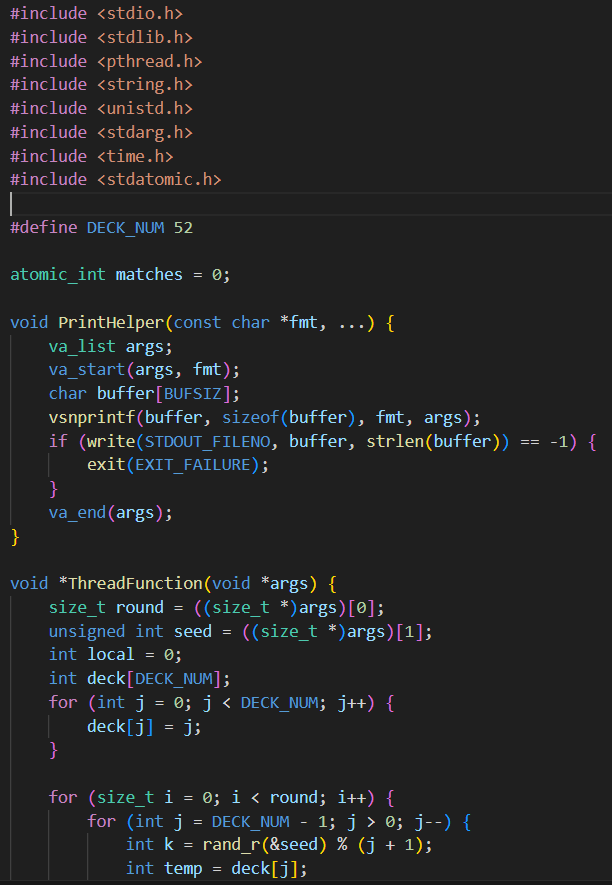
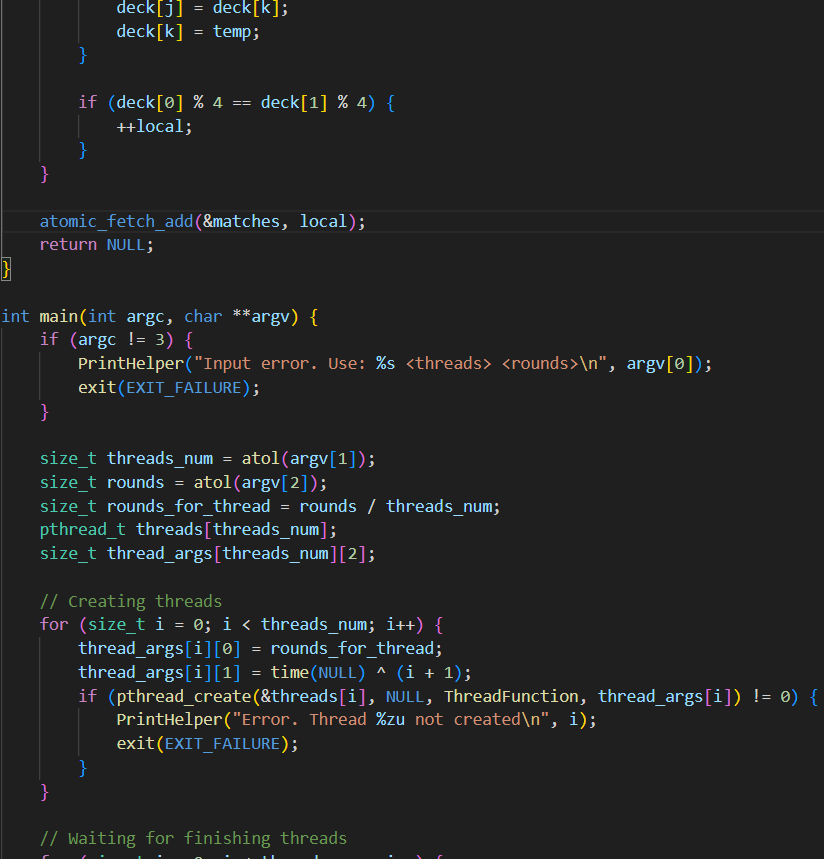
Rounds = 100000000

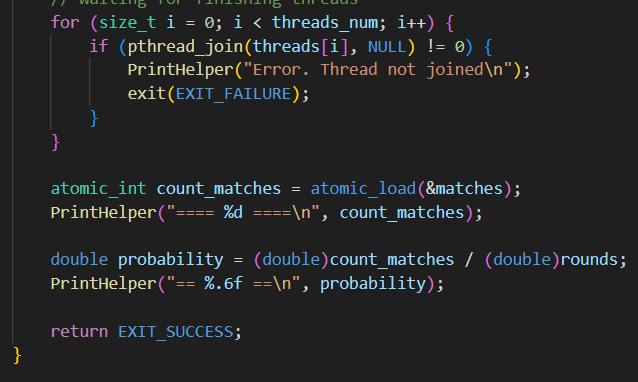
|  |  |  |  |
| --- | --- | --- | --- |
| число потоков | время выполнения, мс | ускорение | Эффективность1 |
| 1 | 17690 | 1,00 | 1,00 |
| 2 | 9214 | 1,91 | 0,995 |
| 3 | 6378 | 2,77 | 0,923 |
| 4 | 4939 | 3,58 | 0,895 |
| 5 | 4332 | 4,08 | 0,816 |
| 6 | 3940 | 4,49 | 0,748 |
| 7 | 3661 | 4,83 | 0,69 |
| 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 |

****

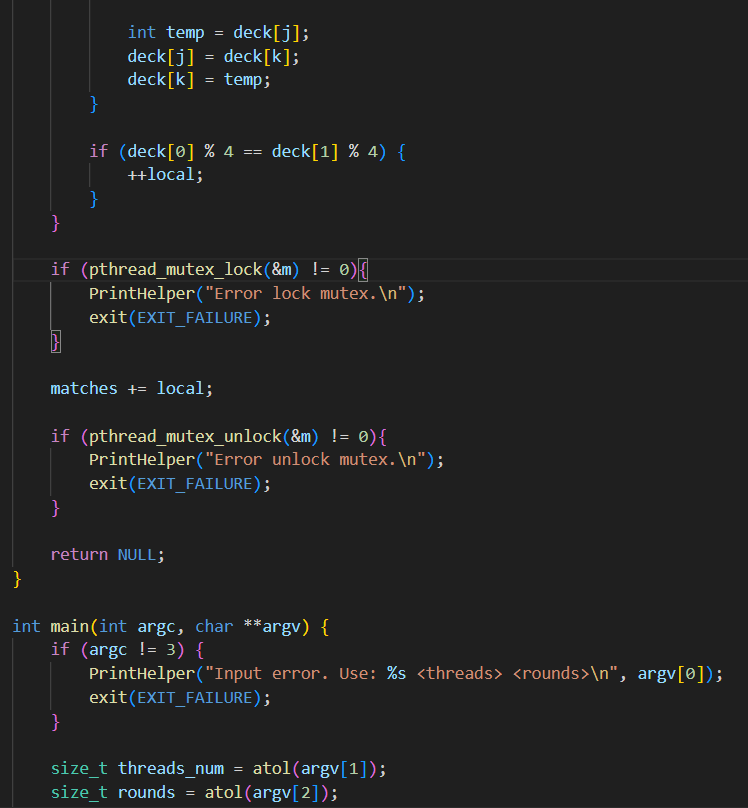
****

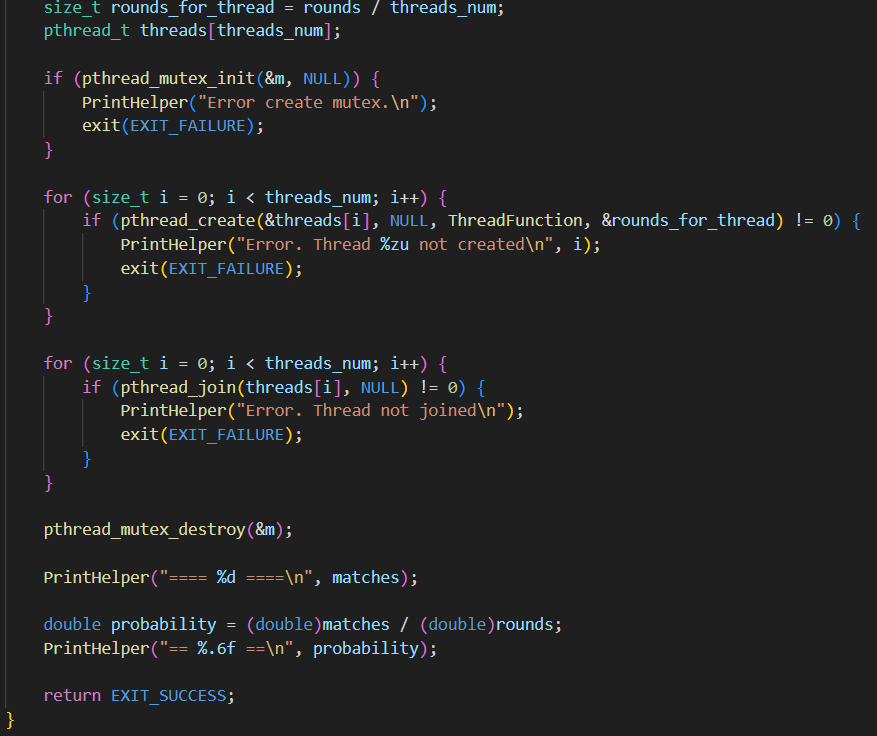
**Код программы**

**main.c (atomic)**



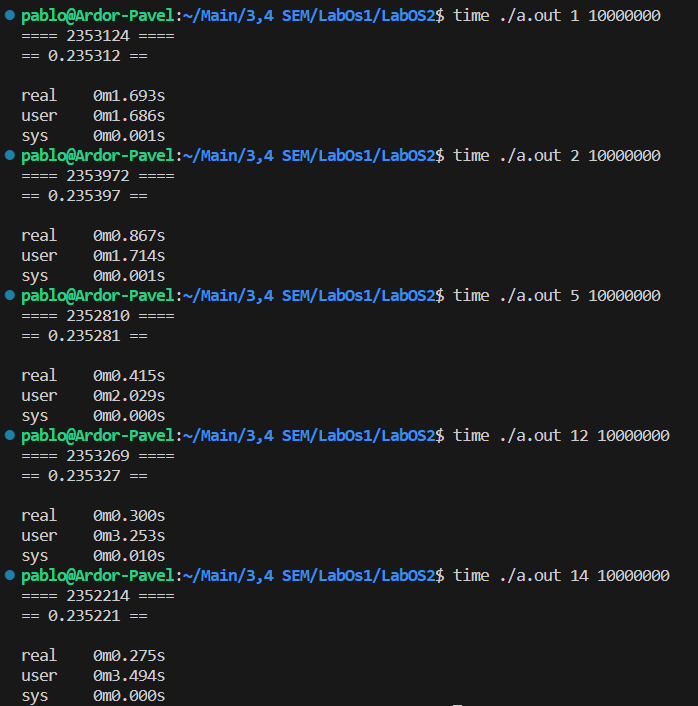
**main\_copy.c (mutex)**



**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\_CLEARTID, 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.