Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

Студент: Носов А.К.

Группа: М8О-206Б-22

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Постановка задачи**

**Вариант 14:** Есть набор 128 битных чисел, записанных в шестнадцатеричном представлении, хранящихся в файле. Необходимо посчитать их среднее арифметическое. Округлить результат до целых. Количество используемой оперативной памяти должно задаваться "ключом".

**Метод решения**

**Используются следующие системные вызовы:**

1. **thread() –** создает новый поток выполнения в программе.
2. **thread.join()** – для блокировки потока.

**Общие сведения о программе**

Для реализации поставленной задачи нам нужны следующие библиотеки: <unistd.h> - для работы с системными вызовами в Linux.

<limits.h> - для определения характеристик общих типов переменных.

<stdlib.h> - для того, чтобы можно было пользоваться функциями, отвечающими за работу с памятью.

<time.h> - для функций, работающих со временем (нужно для строчки srand(time(NULL)) - для генерации рандомных чисел с использованием текущего времени).  
<pthread.h> - для работы с потоками.

<ctype.h> - для классификации и преобразования отдельных символов.

<sys/stat.h> - для доступа к файлам.

<fcntl.h> - для работы с файловым дескриптором.

<inttypes.h> - макросы для использования с функциями printf и scanf.

<string.h> - для использования функций над строками.

Для работы с потоками согласно заданию помимо библиотеки <pthreads.h> я использую такие системные вызовы, как pthread\_create, отвечающий за создание потока, имеющий тип возвращаемого значения int и принимающий 4 аргумента: указатель на поток, атрибуты по умолчанию, указатель на нужную нам функцию и аргументы), а также pthread\_join, отвечающий  
за ожидание завершения потока, имеющий тип возвращаемого значения int и принимающий 2 аргумента: указатель на поток и указатель на указатель в качестве аргумента для хранения возвращаемого значения).

Помимо системных вызовов, связанных с потоками, в моей программе имеются следующие системные вызовы:  
off\_t lseek(...) - устанавливает смещение для файлового дескриптора в значение аргумента offset.

int open(...) - открытие файлового дескриптора.

void exit(...) - выход из процесса с заданным статусом.

int close(...) - закрытие файлового дескриптора.

int write(...) - записывает количество байтов в 3 аргументе из буфера в файл с дескриптором fd.  
Программа собирается и запускается при помощи следующих команд:

gcc lab3.c -pthread -o main

./main thread\_number memory\_amount (пример: ./main 1 72).

**Общий метод и алгоритм решения**

Я создаю две структуры. Первая структура struct Command отвечает за пользовательские данные: количество подающейся оперативной памяти и количество потоков. Вторая стуктура struct Params хранит в себе параметры для одного потока. С запуском программы в файле генерируются числа, и после генерации каждый поток инициализируется начальными данными в функции void initialization(...), принимающей три аргумента: указатель на пользовательскую структуру, указатель на структуру одного потока и количество чисел. Эта же функция распределяет числа по потокам. Далее каждый поток считает свою локальную сумму, в конце работы программы локальные суммы складываются и подсчитывается среднее арифметическое.

**Анализ скорости и эффективности**

Рассмотрим сначала 10000 чисел.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество потоков **p** | Время сортировки с одним потоком **T1** (мкс) | Сортировка с **p** потоками **Tp** (мкс) | Ускорение  (**Sp**=**T1/Tp**) | Эффективность  (**Xp**=Sp/p) |
| 1 | 115504 | 115504 | 1 | 1 |
| 2 | 115504 | 87468 | 1.32 | 0.66 |
| 3 | 115504 | 79805 | 1.45 | 0.48 |
| 4 | 115504 | 75855 | 1.52 | 0.38 |
| 5 | 115504 | 81048 | 1.43 | 0.29 |
| 6 | 115504 | 69663 | 1.66 | 0.28 |
| 7 | 115504 | 76898 | 1.5 | 0.21 |
| 8 | 115504 | 83455 | 1.38 | 0.17 |
| 9 | 115504 | 74030 | 1.56 | 0.17 |
| 10 | 115504 | 65547 | 1.76 | 0.18 |

Здесь видно, что эффективность понижается с увеличением числа потоков, но плохо просматривается закономерность в ускорении.

Возьмём массив из 50000 тысяч чисел.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество потоков **p** | Время сортировки с одним потоком **T1** (мкс) | Время сортировки с одним потоком **T1** (мкс) | Ускорение  (**Sp**=**T1/Tp**) | Эффективность  (**Xp**=Sp/p) |
| 1 | 590869 | 590869 | 1 | 1 |
| 2 | 590869 | 299094 | 1,98 | 0.99 |
| 3 | 590869 | 364958 | 1,62 | 0.54 |
| 4 | 590869 | 366851 | 1,61 | 0.40 |
| 5 | 590869 | 353109 | 1,67 | 0.33 |
| 6 | 590869 | 310636 | 1.90 | 0.32 |
| 7 | 590869 | 395236 | 1.49 | 0.21 |
| 8 | 590869 | 302383 | 1.95 | 0.24 |
| 9 | 590869 | 337434 | 1.75 | 0.19 |
| 10 | 590869 | 375124 | 1.58 | 0.16 |
| 100 | 590869 | 280406 | 2.11 | 0.02 |

Здесь видно, что при большом количестве потоков ускорение падает, и всё же погрешность ещё велика.

Последней возьмём последовательность из 100000 чисел.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество потоков **p** | Время сортировки с одним потоком **T1** (мкс) | Сортировка с **p** потоками **Tp** (мкс) | Ускорение  (**Sp**=**T1/Tp**) | Эффективность  (**Xp**=Sp/p) |
| 1 | 960749 | 960748 | 1 | 1 |
| 2 | 960748 | 717900 | 1,34 | 0,67 |
| 3 | 960748 | 897206 | 1,07 | 0,36 |
| 4 | 960748 | 663090 | 1,45 | 0,36 |
| 5 | 960748 | 743881 | 1,29 | 0,25 |
| 6 | 960748 | 588472 | 1,63 | 0,27 |
| 7 | 960748 | 663981 | 1,45 | 0,21 |
| 8 | 960748 | 659101 | 1,46 | 0,18 |
| 9 | 960748 | 694577 | 1,38 | 0,15 |
| 10 | 960748 | 668109 | 1,44 | 0,14 |
| 15 | 960748 | 813362 | 1,18 | 0,08 |
| 1000 | 960748 | 783225 | 1,23 | 0,001 |
| 10000 | 960748 | 779213 | 1,23 | 0,0001 |

Как мы видим, при большом количестве потоков эффективность сильно снижается. На большом количестве данных не очень большое ускорение.

**Исходный код**

**lab2.h**

#include <stdio.h>

#include <unistd.h>

#include <limits.h>

#include <stdlib.h>

#include <time.h>

#include <pthread.h>

#include <ctype.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <inttypes.h>

#include <string.h>

#define DEC\_SIZE 40

#define NUMBER\_SIZE 32

#define FILE\_SIZE 100

#define TERMINAL\_NULL '\0'

typedef unsigned \_\_int128 int128\_t;

char \*file\_name = "file.txt";

void generate()

{

int fd = open(file\_name, O\_RDWR| O\_CREAT, 0666);

char array[NUMBER\_SIZE];

srand(time(NULL));

for (int i = 0; i < FILE\_SIZE; ++i) {

for (int i = 0; i < NUMBER\_SIZE; ++i) {

if (((int) rand()) % 2 == 0) {

array[i] = '0' + (((int) rand()) % 10);

} else {

array[i] = 'A' + (((int) rand()) % 6);

}

}

write(fd, &array, NUMBER\_SIZE);

write(fd, "\n", 1);

}

close(fd);

}

void print(int128\_t num)

{

printf("DEC: \n");

char array[DEC\_SIZE + 1];

int i;

for (i = 0; i < DEC\_SIZE; ++i) {

array[i] = '0';

}

array[DEC\_SIZE] = TERMINAL\_NULL;

for (i = DEC\_SIZE - 1; num > 0; --i) {

array[i] = (int) (num % 10) + '0'; //представляем в коде аски и заполняем массив с конца

num /= 10;

}

printf("%s\n", &array[i + 1]);

}

int number\_checker(char \*s)

{

return (\*s >= '0' && \*s <= '9');

}

int hexdexconvert(char \*s) {

if(\*s == 'a')

return 10;

if(\*s == 'b')

return 11;

if(\*s == 'c')

return 12;

if(\*s == 'd')

return 13;

if(\*s == 'e')

return 14;

if(\*s == 'f')

return 15;

return 0;

}

int128\_t to128bit(char \*s)

{

int128\_t num = 0;

while (\*s) {

if (number\_checker(s))

num = num \* 16 + (\*s - '0');

else {

int b = hexdexconvert(s);

num = num \* 16 + b;

}

++s;

}

return num;

}

struct Command {

int memory;

int number\_of\_threads;

};

struct Params {

long long num\_count;

int128\_t localsum;

off\_t start\_pos;

long long thread\_count\_controller;

};

void initialization(struct Params \*ptr, struct Command \*command, long long num\_count)

{

ptr[0].num\_count = num\_count;

ptr[0].localsum = 0;

ptr[0].start\_pos = 0;

ptr[0].thread\_count\_controller = (num\_count / command->number\_of\_threads);

for (int i = 1; i < command->number\_of\_threads; ++i) {

ptr[i].num\_count = num\_count;

ptr[i].localsum = 0;

ptr[i].start\_pos = i \* (ptr[i - 1].thread\_count\_controller \* (NUMBER\_SIZE + 1));

ptr[i].thread\_count\_controller = ptr[i-1].thread\_count\_controller;

}

ptr[command->number\_of\_threads - 1].thread\_count\_controller += num\_count % command->number\_of\_threads;

}

void parse\_command\_line(int argc, char\* argv[], struct Command \*command)

{

if (argc != 3) {

fprintf(stderr, "%s\n","Usage: ThreadsNumber, Memory");

exit(EXIT\_FAILURE);

}

command->number\_of\_threads = atoi(argv[1]);

command->memory = atoi(argv[2]);

}

void \*thread\_function(void \*a)

{

struct Params \*ptr = (struct Params\*)a;

char array[NUMBER\_SIZE + 1];

char c;

int fd = open(file\_name, O\_RDWR| O\_CREAT, 0666);

lseek(fd, ptr->start\_pos, SEEK\_SET);

for (int i = 0; i < ptr->thread\_count\_controller; ++i)

{

read(fd, array, NUMBER\_SIZE);

array[NUMBER\_SIZE] = TERMINAL\_NULL;

int128\_t s;

s = to128bit(array);

s /= ptr->num\_count;

ptr->localsum += s;

read(fd, &c, 1);

}

close(fd);

return 0;

}

int main(int argc, char\* argv[])

{

struct Command command;

parse\_command\_line(argc, argv, &command);

if (command.number\_of\_threads \* sizeof(struct Params) + command.number\_of\_threads \* sizeof(pthread\_t) > command.memory) {

fprintf(stderr, "%s\n", "Too much threads for this amount of memory");

exit(EXIT\_FAILURE);

}

pthread\_t \*thread\_id = (pthread\_t\*)malloc(command.number\_of\_threads \* sizeof(pthread\_t));

struct Params \*params = (struct Params\*)malloc(command.number\_of\_threads \* sizeof(struct Params));

generate();

int fd = open(file\_name, O\_RDWR| O\_CREAT, 0666);

long long size = lseek(fd, 0, SEEK\_END);

close(fd);

long long num\_count = size / (NUMBER\_SIZE + 1);

initialization(params, &command, num\_count);

for (int i = 0; i < command.number\_of\_threads; ++i) {

pthread\_create(&thread\_id[i], NULL, thread\_function, (void \*) &params[i]);

}

for (int j = 0; j < command.number\_of\_threads; ++j) {

pthread\_join(thread\_id[j], NULL);

}

int128\_t res = 0;

for (int i = 0; i < command.number\_of\_threads; ++i) {

res += params[i].localsum;

}

print(res);

free(thread\_id);

free(params);

return 0;

}

**Strace**

execve("./lab2", ["./lab2", "1", "56"], 0x7fffc3fb8b40 /\* 87 vars \*/) = 0

brk(NULL) = 0x55b6e54f8000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffe6c9d8dc0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fc02c4e1000

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=70887, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 70887, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fc02c4cf000

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\0\302\211\332Pq\2439\235\350\223\322\257\201\326\243\f"..., 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) = 0x7fc02c200000

mprotect(0x7fc02c228000, 2023424, PROT\_NONE) = 0

mmap(0x7fc02c228000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7fc02c228000

mmap(0x7fc02c3bd000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7fc02c3bd000

mmap(0x7fc02c416000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7fc02c416000

mmap(0x7fc02c41c000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fc02c41c000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fc02c4cc000

arch\_prctl(ARCH\_SET\_FS, 0x7fc02c4cc740) = 0

set\_tid\_address(0x7fc02c4cca10) = 34884

set\_robust\_list(0x7fc02c4cca20, 24) = 0

rseq(0x7fc02c4cd0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7fc02c416000, 16384, PROT\_READ) = 0

mprotect(0x55b6e354c000, 4096, PROT\_READ) = 0

mprotect(0x7fc02c51b000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7fc02c4cf000, 70887) = 0

getrandom("\x36\xde\x97\x41\xc6\xf5\xea\x2b", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55b6e54f8000

brk(0x55b6e5519000) = 0x55b6e5519000

openat(AT\_FDCWD, "file.txt", O\_RDWR|O\_CREAT, 0666) = 3

write(3, "30FEE6F6EC6FEBEE79E2293FC65EFDBB", 32) = 32

write(3, "\n", 1) = 1

write(3, "9BB7CFCDDB6CBB26F3261E2DC666DBFA", 32) = 32

write(3, "\n", 1) = 1

write(3, "FBEAB7D9DDDA0F8B35CAF06FEEE3F724", 32) = 32

write(3, "\n", 1) = 1

write(3, "AE3161D27D1DD9CC49B8CE7EEBA88F28", 32) = 32

write(3, "\n", 1) = 1

write(3, "EEC71CC60C50C34B90ED87AD2BAC1AC0", 32) = 32

write(3, "\n", 1) = 1

write(3, "E810EA4335BFC4E06EBFEBF5EC21EC22", 32) = 32

write(3, "\n", 1) = 1

write(3, "387D6D531A3DCF9BCBED82FD3A2EEF53", 32) = 32

write(3, "\n", 1) = 1

write(3, "F2F8C3A0A94B5A3FCCF339CA6D367EEE", 32) = 32

write(3, "\n", 1) = 1

write(3, "625C3FBEADDBEAA1CBEFC1FC0BDAA0E2", 32) = 32

write(3, "\n", 1) = 1

write(3, "BE034652CDACEC7914C96C29414F0CF4", 32) = 32

write(3, "\n", 1) = 1

write(3, "878AAB94FAFBFB7BB397788AE4FB3918", 32) = 32

write(3, "\n", 1) = 1

write(3, "0E9A553B951329EA0648BB4ED46BF6F1", 32) = 32

write(3, "\n", 1) = 1

write(3, "F7C4B2354ACCF0702D9CD562CA4240E9", 32) = 32

write(3, "\n", 1) = 1

write(3, "41CEDAC89064D4B8AE6EE24E9C992AB6", 32) = 32

write(3, "\n", 1) = 1

write(3, "9CA241DBAF72D3EFEE85353132407DC7", 32) = 32

write(3, "\n", 1) = 1

write(3, "BDA2CF4EB868BBB816C4B9BD1ED51105", 32) = 32

write(3, "\n", 1) = 1

write(3, "EF3B989D4ACDB4AA6DB55B699BC5DCC6", 32) = 32

write(3, "\n", 1) = 1

write(3, "2FF7052919B871AB873EBB581D6ABD38", 32) = 32

write(3, "\n", 1) = 1

write(3, "B55970D30C2358524AFAF89AA9E13A76", 32) = 32

write(3, "\n", 1) = 1

write(3, "65BA43EC4D903AB5C3EA6D4A07CC4919", 32) = 32

write(3, "\n", 1) = 1

write(3, "C56C6A8328B346EF275AB5E8A42E34BD", 32) = 32

write(3, "\n", 1) = 1

write(3, "D38BB819CB4DFAAEC711EDE999A0D2DE", 32) = 32

write(3, "\n", 1) = 1

write(3, "FB73A902EE10019B4FF5F5C15BCD9CA9", 32) = 32

write(3, "\n", 1) = 1

write(3, "D25A3FF2EE4AEAD404C1CBBAA521594A", 32) = 32

write(3, "\n", 1) = 1

write(3, "241DC01FDDFE7AF7CDB8DF0ECFCDAEB8", 32) = 32

write(3, "\n", 1) = 1

write(3, "FB4CAF8E7DC73E72EB0FB5AAAE2DBE79", 32) = 32

write(3, "\n", 1) = 1

write(3, "FDE4D5FC006EA16BEA3E7B8E0F2DB1B4", 32) = 32

write(3, "\n", 1) = 1

write(3, "B4B35DAD42DC8BBFCEDA2BC3F853D789", 32) = 32

write(3, "\n", 1) = 1

write(3, "7F9ECAB247E7C14CE3BB2FFAFDF2EE65", 32) = 32

write(3, "\n", 1) = 1

write(3, "C85EBA04BC72D521ADFE86937D7B18CC", 32) = 32

write(3, "\n", 1) = 1

write(3, "1F0C66BB67D016C2CB1BFC9BA3CAB5AE", 32) = 32

write(3, "\n", 1) = 1

write(3, "2FE4344E5F2BF33A1DD1E93AA334FB8E", 32) = 32

write(3, "\n", 1) = 1

write(3, "F7698F2DB1EACEA4CA4F08FEBE79DAAB", 32) = 32

write(3, "\n", 1) = 1

write(3, "4F05AAF45888D9F85CBAD40E5C559622", 32) = 32

write(3, "\n", 1) = 1

write(3, "DF8BCF243E3A434EABC2890CA32A4E90", 32) = 32

write(3, "\n", 1) = 1

write(3, "FA8C1CBCFBC5130E64528DCCA5E9E8D9", 32) = 32

write(3, "\n", 1) = 1

write(3, "6C5AE61D39F2F20AAD8B70DD61EAAC67", 32) = 32

write(3, "\n", 1) = 1

write(3, "B3C3AEBBB8071C45A364C4C4C52A8C5C", 32) = 32

write(3, "\n", 1) = 1

write(3, "DAF6CA10AAA9FD4CA60F61ACA8FA2B5B", 32) = 32

write(3, "\n", 1) = 1

write(3, "BBEF79F867DAABA9C25D66E3B0343FA6", 32) = 32

write(3, "\n", 1) = 1

write(3, "302F0D5D17AB67764C0A67964DA1B946", 32) = 32

write(3, "\n", 1) = 1

write(3, "B7A28CBD0FB44ED01ED3B412FC336FEA", 32) = 32

write(3, "\n", 1) = 1

write(3, "A06BF908D3B9E039EB6F3CD67D094ECB", 32) = 32

write(3, "\n", 1) = 1

write(3, "61129F1F98BBDBA7845A65B649C1B4CD", 32) = 32

write(3, "\n", 1) = 1

write(3, "20FDD807F2DE96DAB25703C8C6C0C93F", 32) = 32

write(3, "\n", 1) = 1

write(3, "B9CF71CDAA074699C49A005DE92FBBB4", 32) = 32

write(3, "\n", 1) = 1

write(3, "E152C4BB161BD4400ADFDFA74E5D1CF1", 32) = 32

write(3, "\n", 1) = 1

write(3, "E0FD43E76AEFAEEC5B89C69DCA1DCD6A", 32) = 32

write(3, "\n", 1) = 1

write(3, "04CAC5D6A9F2BEB5C6F0ACE44FDF9C7B", 32) = 32

write(3, "\n", 1) = 1

write(3, "55ECAA1E0D9014CBB49DCCA3EBBDCA2B", 32) = 32

write(3, "\n", 1) = 1

write(3, "185DACE09F433CC1ADAEC064FADE10CB", 32) = 32

write(3, "\n", 1) = 1

write(3, "31BA462CC31C982EF8DBDC61C1FCA39D", 32) = 32

write(3, "\n", 1) = 1

write(3, "9DA8800F06C40CDB2BF1ACED2A38A6F9", 32) = 32

write(3, "\n", 1) = 1

write(3, "DC5CC4B5DDAA7B983804690D8F4C4AE7", 32) = 32

write(3, "\n", 1) = 1

write(3, "F6C9EDE3913C13BA19A8DAAEFB0AC6DF", 32) = 32

write(3, "\n", 1) = 1

write(3, "53457FEFE7C28FE3E34097B455AF0548", 32) = 32

write(3, "\n", 1) = 1

write(3, "8CCFD19E1FB1E882B2DD715EFA17EFFA", 32) = 32

write(3, "\n", 1) = 1

write(3, "E8FC3BECF52ACE79AED7118DCD2D9C5F", 32) = 32

write(3, "\n", 1) = 1

write(3, "DE64C15FCA4CBF80A729CBCF8C40DE69", 32) = 32

write(3, "\n", 1) = 1

write(3, "0ABAAC8E21DE6626B0879F7ECACDF585", 32) = 32

write(3, "\n", 1) = 1

write(3, "CB93301EE8F4AF39966CFAE02F1BB940", 32) = 32

write(3, "\n", 1) = 1

write(3, "CF8B0B3F40CAFBF2DDEDC96298B84DBF", 32) = 32

write(3, "\n", 1) = 1

write(3, "265F6A1269DDB21DCDAF2EDEDE9B141B", 32) = 32

write(3, "\n", 1) = 1

write(3, "EBA525BD5D54638593DBDA2ADB2A0C28", 32) = 32

write(3, "\n", 1) = 1

write(3, "A3B89A01AF061EBFB94E08AA00EFC3F6", 32) = 32

write(3, "\n", 1) = 1

write(3, "6E98E00D3BAE7B96B84852BD9BC0FDDE", 32) = 32

write(3, "\n", 1) = 1

write(3, "D7BD8FF85FFAFA87DC4A5C58DADC04E6", 32) = 32

write(3, "\n", 1) = 1

write(3, "FDECE10DDE82E0C8B53F3ECAEDBA216A", 32) = 32

write(3, "\n", 1) = 1

write(3, "9CCFACC435FBD652F2DC836E37D2E7EA", 32) = 32

write(3, "\n", 1) = 1

write(3, "B018EDB147C62BA0EE376FB3EA3ACCFA", 32) = 32

write(3, "\n", 1) = 1

write(3, "20BBE0AC6AA90DC34EDEE18D6C0D21AD", 32) = 32

write(3, "\n", 1) = 1

write(3, "EAFBD7F4841EEE03B61F3ED9CDFEBFFA", 32) = 32

write(3, "\n", 1) = 1

write(3, "3F6831C673379E00BD0CAC9D1369DFD4", 32) = 32

write(3, "\n", 1) = 1

write(3, "FFFDEBC9F2DF80C24DC44F0CF47FAE9A", 32) = 32

write(3, "\n", 1) = 1

write(3, "C4A50F8DFA7DABE1CC6B229A48DEA8EC", 32) = 32

write(3, "\n", 1) = 1

write(3, "76BB3A2022B84F8CBA6A4A9F9FF752FF", 32) = 32

write(3, "\n", 1) = 1

write(3, "0C7CAECFEEDB1392ABE815FCEFA926BA", 32) = 32

write(3, "\n", 1) = 1

write(3, "A860FE443D525111AF7B7FDEEDEDFDB4", 32) = 32

write(3, "\n", 1) = 1

write(3, "D1531679D8F2BA110362A3CA53E6F3C5", 32) = 32

write(3, "\n", 1) = 1

write(3, "E16F6B8CA082ACA66FE91BCD5FAB0B7F", 32) = 32

write(3, "\n", 1) = 1

write(3, "09E71302EF66F3F903E576B4CB7BDEBD", 32) = 32

write(3, "\n", 1) = 1

write(3, "77E9CACA8A135B82CCB65DE7942DC515", 32) = 32

write(3, "\n", 1) = 1

write(3, "D0CFC9688E2BCBDE1BC47AA19539F23F", 32) = 32

write(3, "\n", 1) = 1

write(3, "873877CC5E5AF9F5BF9E5A02F6312D3F", 32) = 32

write(3, "\n", 1) = 1

write(3, "8AC76C1463C51EA315E34FDEFAF7D8CF", 32) = 32

write(3, "\n", 1) = 1

write(3, "B8867CE8DC114B592E3FCCF49A74DF73", 32) = 32

write(3, "\n", 1) = 1

write(3, "DAEAC6E19E7DA7612F4B3DA1E2BCA52E", 32) = 32

write(3, "\n", 1) = 1

write(3, "4BE7AF79CBBFF5107D4930BEFB39866E", 32) = 32

write(3, "\n", 1) = 1

write(3, "4A7CA6C0CBF748D9BE639FC09FB3ECBB", 32) = 32

write(3, "\n", 1) = 1

write(3, "2AD73FFD61D7FF96DE37162E66AC0EFB", 32) = 32

write(3, "\n", 1) = 1

write(3, "6FF78C9FFCC380DAEC1EDA0DAA12C1BA", 32) = 32

write(3, "\n", 1) = 1

write(3, "00D298BCEF75E632E6AE51C672850E97", 32) = 32

write(3, "\n", 1) = 1

write(3, "C4F359A5C9289A7747A3AEB0A96E6A10", 32) = 32

write(3, "\n", 1) = 1

write(3, "CC1EA4AE7AFBC5CA40F79FA2B938CC34", 32) = 32

write(3, "\n", 1) = 1

write(3, "CD325E19A75DD825B1EF4C5F88CAF971", 32) = 32

write(3, "\n", 1) = 1

write(3, "19DBCB58B0F8BB5206DAD52BDAF89EDF", 32) = 32

write(3, "\n", 1) = 1

write(3, "FA950CAC8DDCA2BE9286D8B9F4D2B051", 32) = 32

write(3, "\n", 1) = 1

write(3, "8CD286CA8BDF5F277FAF3C14FAAFB5C9", 32) = 32

write(3, "\n", 1) = 1

write(3, "6F6066B217EEDA6E0CD1DE9295FE8870", 32) = 32

write(3, "\n", 1) = 1

write(3, "77F49A124DEA79BF29A7DB9F836EACBF", 32) = 32

write(3, "\n", 1) = 1

close(3) = 0

openat(AT\_FDCWD, "file.txt", O\_RDWR|O\_CREAT, 0666) = 3

lseek(3, 0, SEEK\_END) = 3300

close(3) = 0

rt\_sigaction(SIGRT\_1, {sa\_handler=0x7fc02c291870, sa\_mask=[], sa\_flags=SA\_RESTORER|SA\_ONSTACK|SA\_RESTART|SA\_SIGINFO, sa\_restorer=0x7fc02c242520}, 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) = 0x7fc02b9ff000

mprotect(0x7fc02ba00000, 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=0x7fc02c1ff910, parent\_tid=0x7fc02c1ff910, exit\_signal=0, stack=0x7fc02b9ff000, stack\_size=0x7fff00, tls=0x7fc02c1ff640} => {parent\_tid=[34885]}, 88) = 34885

rt\_sigprocmask(SIG\_SETMASK, [], NULL, 8) = 0

futex(0x7fc02c1ff910, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 34885, NULL, FUTEX\_BITSET\_MATCH\_ANY) = 0

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

write(1, "DEC: \n", 6DEC:

) = 6

write(1, "46541449712355940947844612565457"..., 3946541449712355940947844612565457047589

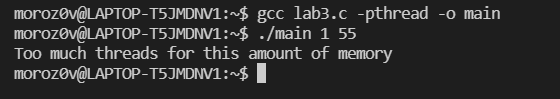
) = 39

exit\_group(0) = ?

+++ exited with 0 +++

**Демонстрация работы программы**Тест 1.

gcc lab3.c –pthread -o main

./main 1 55Тест 2.  
./main 1 56  


Тест 3.

./main 10 5600



**Выводы**Данная лабораторная работа помогла мне успешно ознакомиться с тем, как устроены потоки в Linux. Во время выполнения своего задания я изучил особенности системных вызовов и узнал многие тонкости работы с потоками.