МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЕТ

по лабораторной работе № 1

по дисциплине «Операционные системы»

ТЕМА: «Управление файловой системой»

Студент гр. 3311 Баймухамедов Р. Р.

Преподаватель Тимофеев А. В.

Санкт-Петербург

2025

Цель работы
Приложение должно копировать существующий файл в новый файл, «одновременно» выполняя n перекрывающихся операций ввода-вывода (механизм APC) блоками данных кратными размеру кластера.

Задание
Создайте консольное приложение, которое выполняет: − открытие/создание файлов. Измерьте продолжительности выполнения операции копирования файла. Проверьте его работоспособность на копировании файлов разного размера для ситуации с перекрывающимся выполнением одной операции ввода и одной операции вывода. Определите оптимальный размер блока данных, при котором скорость копирования наибольшая. Произведите замеры времени выполнения приложения для разного числа перекрывающихся операций ввода и вывода. По результатам измерений постройте график зависимости и определите число перекрывающихся операций ввода и вывода, при котором достигается наибольшая скорость копирования файла.

Постановка задачи и описание решения
Для выполнения данной лабораторной работы необходимо разработать консольное приложение, которое

- Размер блока данных – определить, как размер копируемого блока влияет на скорость копирования.
- Число перекрывающихся операций ввода-вывода – найти оптимальное количество одновременных операций, при котором достигается максимальная скорость копирования.

В рамках эксперимента необходимо:

- Провести копирование файлов разного размера при фиксированном числе перекрывающихся и построить график зависимости скорости копирования от размера блока данных.
- Провести копирование файлов при разном числе перекрывающихся операций (1, 2, 4, 8, 12, 16) и построить график зависимости скорости копирования от их количества.
- Определить оптимальные параметры копирования, при которых достигается наибольшая скорость.

Примеры выполнения программы
Примеры работоспособности консольного приложения продемонстрированы ниже

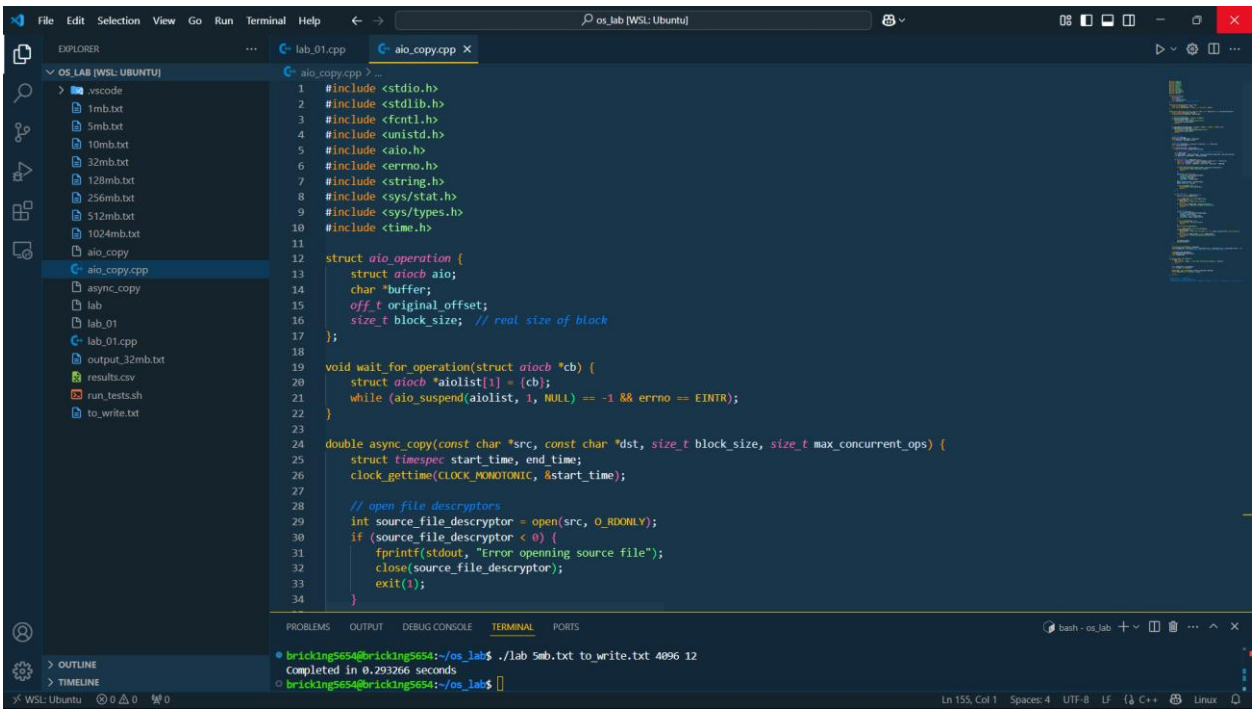
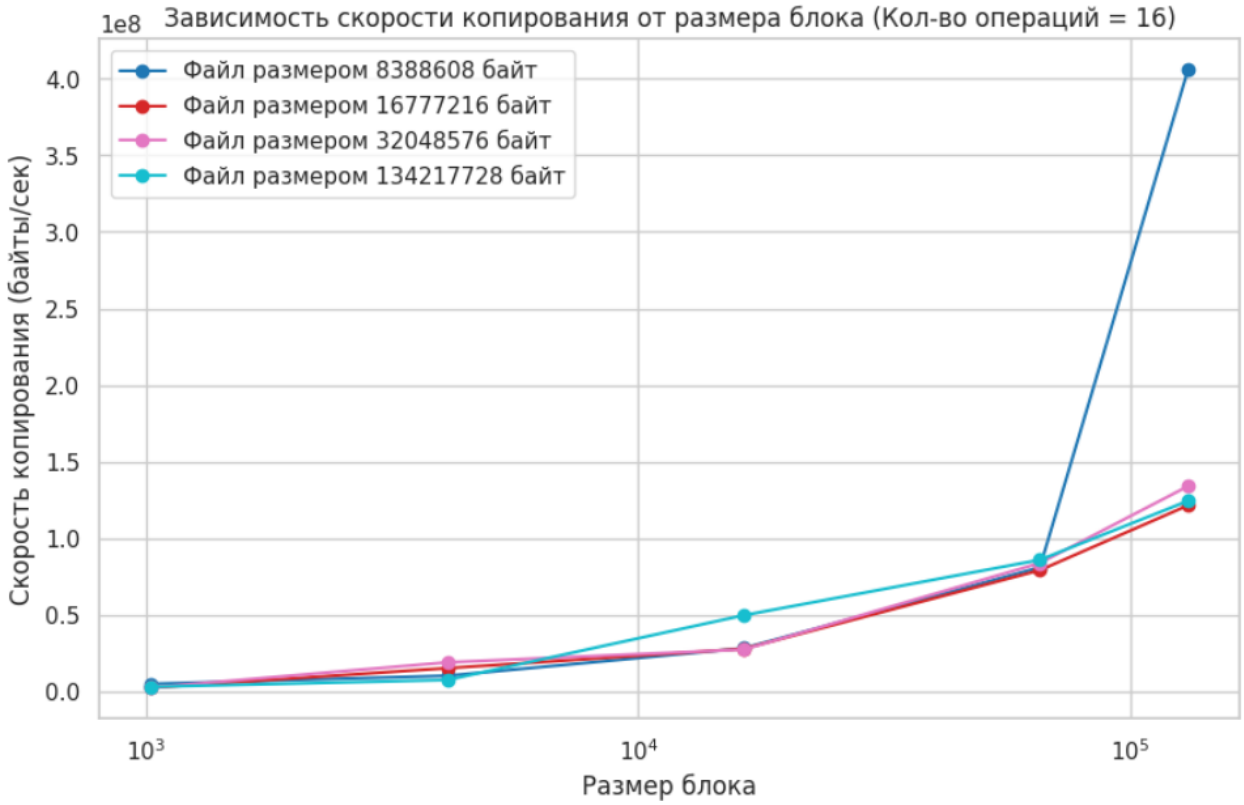
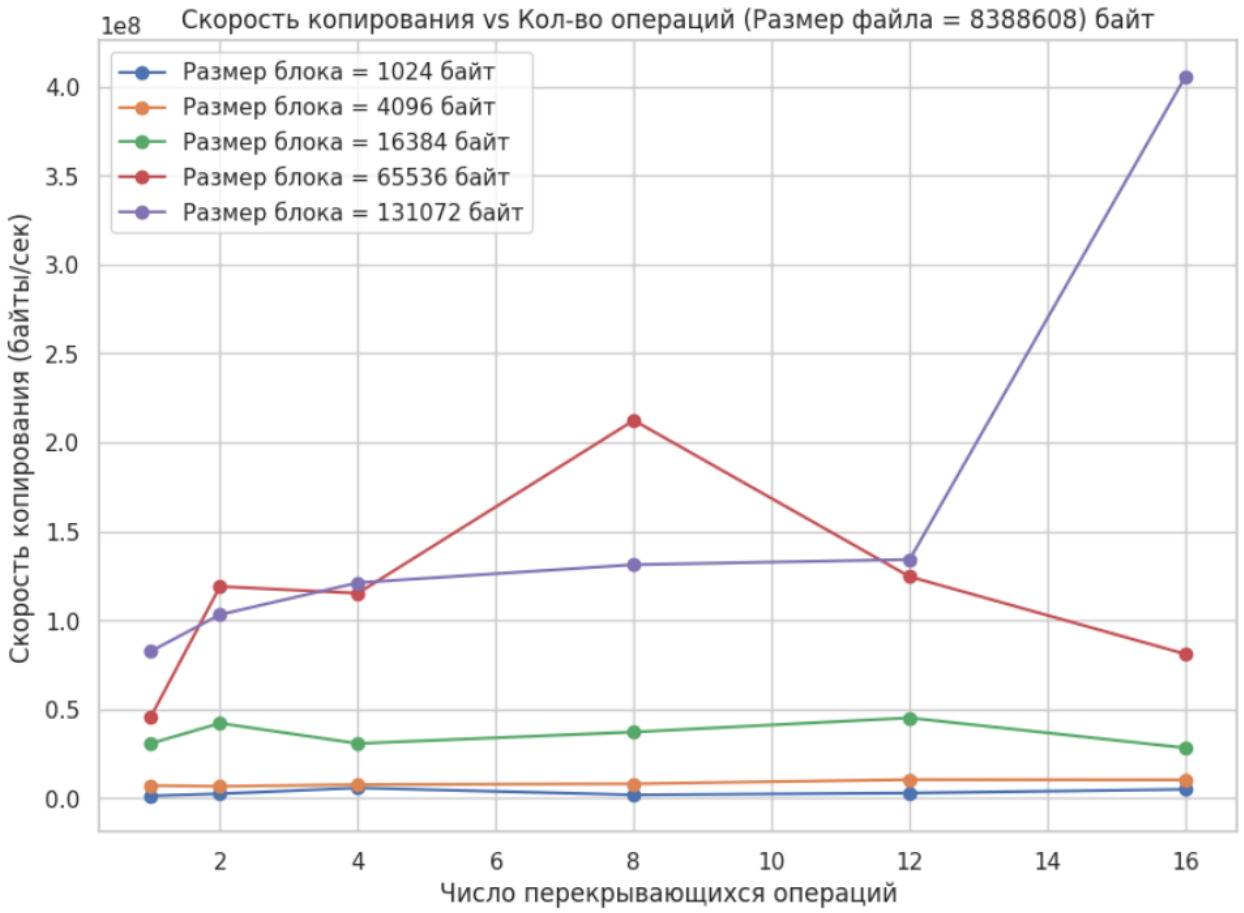
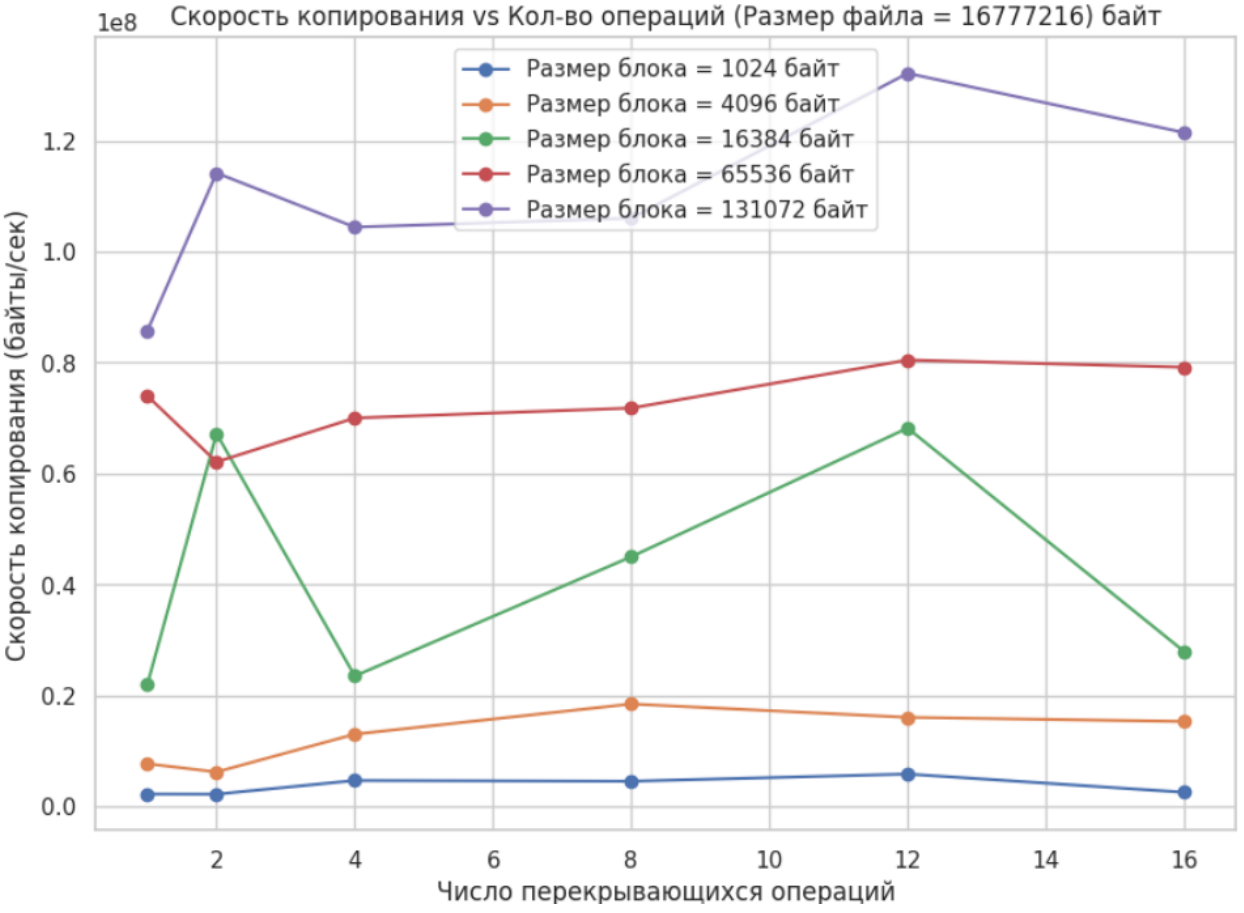


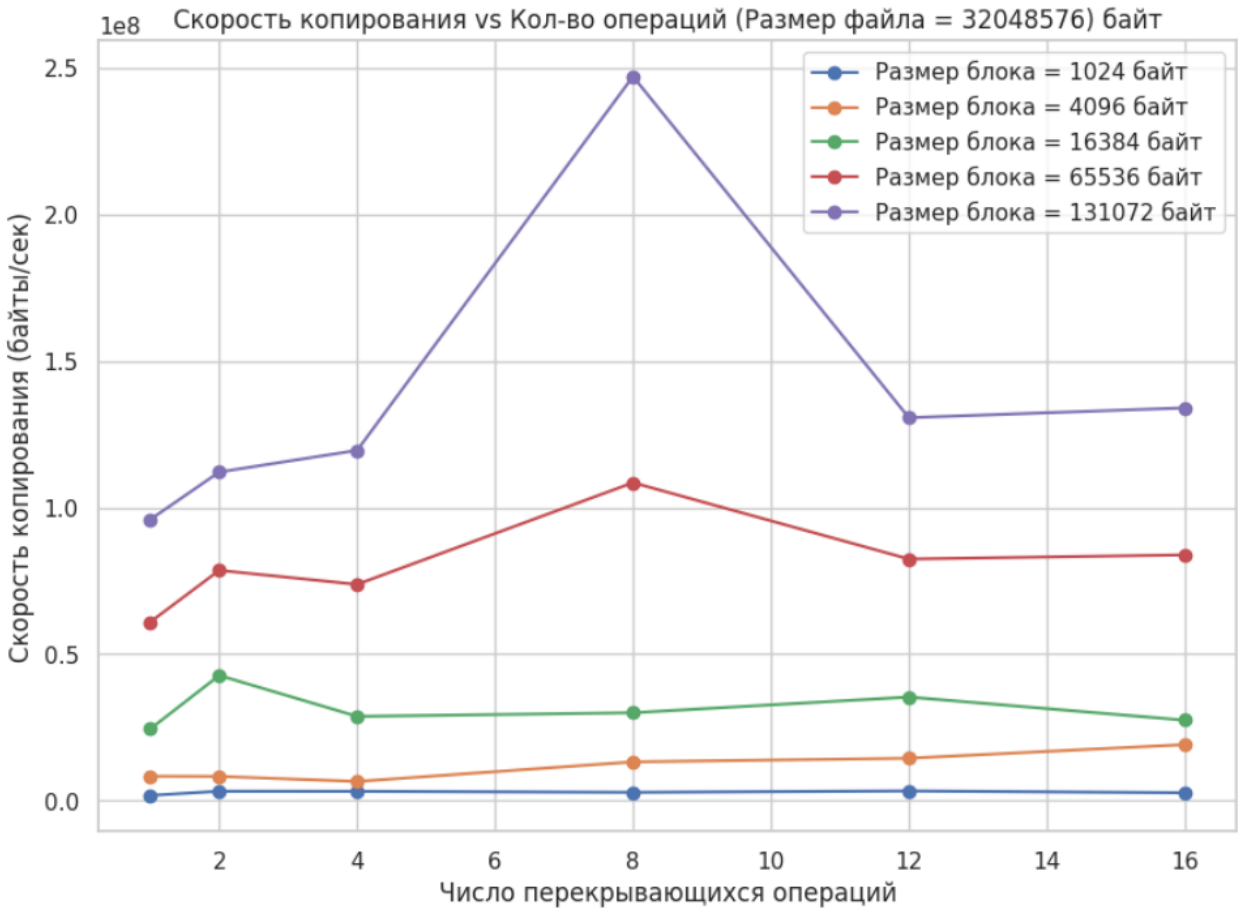
График зависимости скорости копирования от размера блока данных:

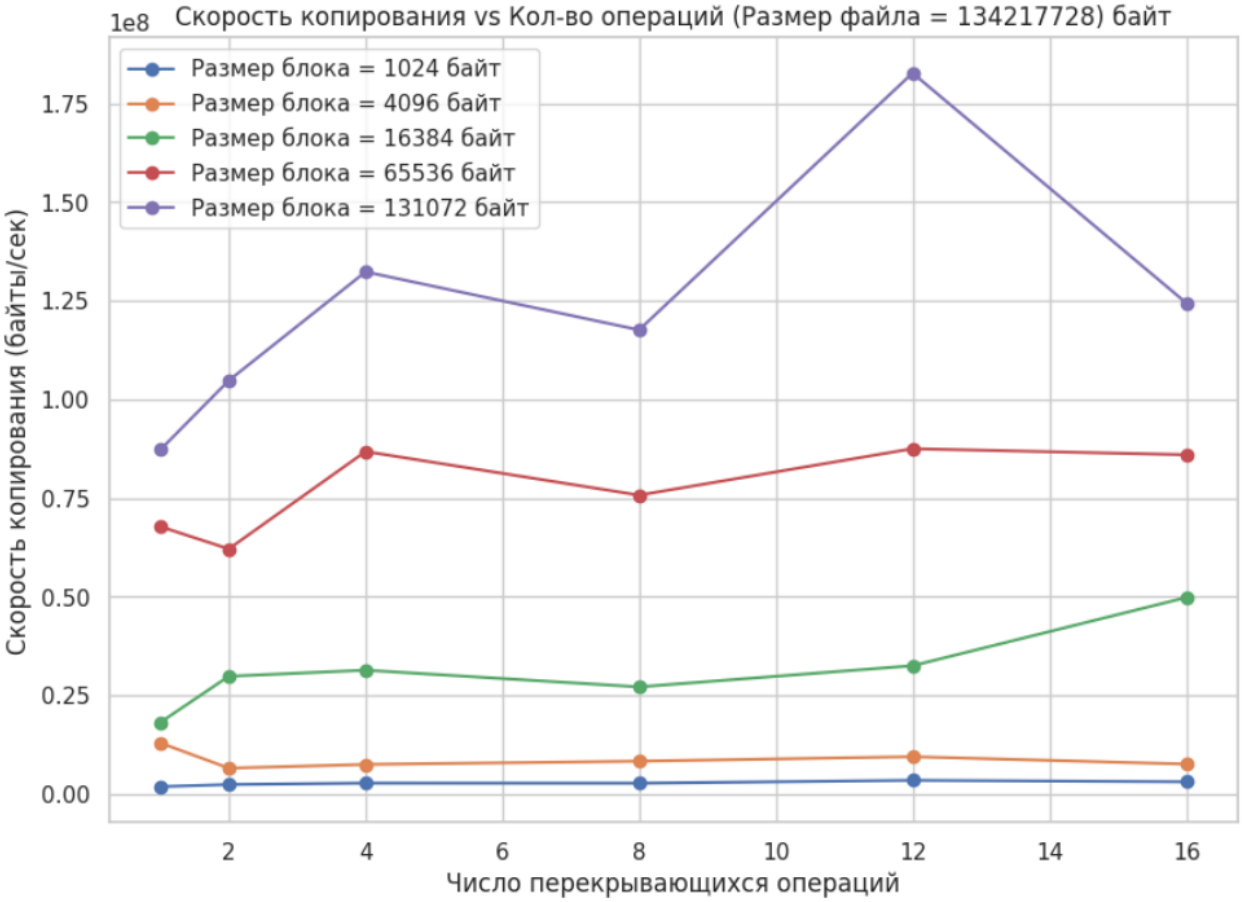


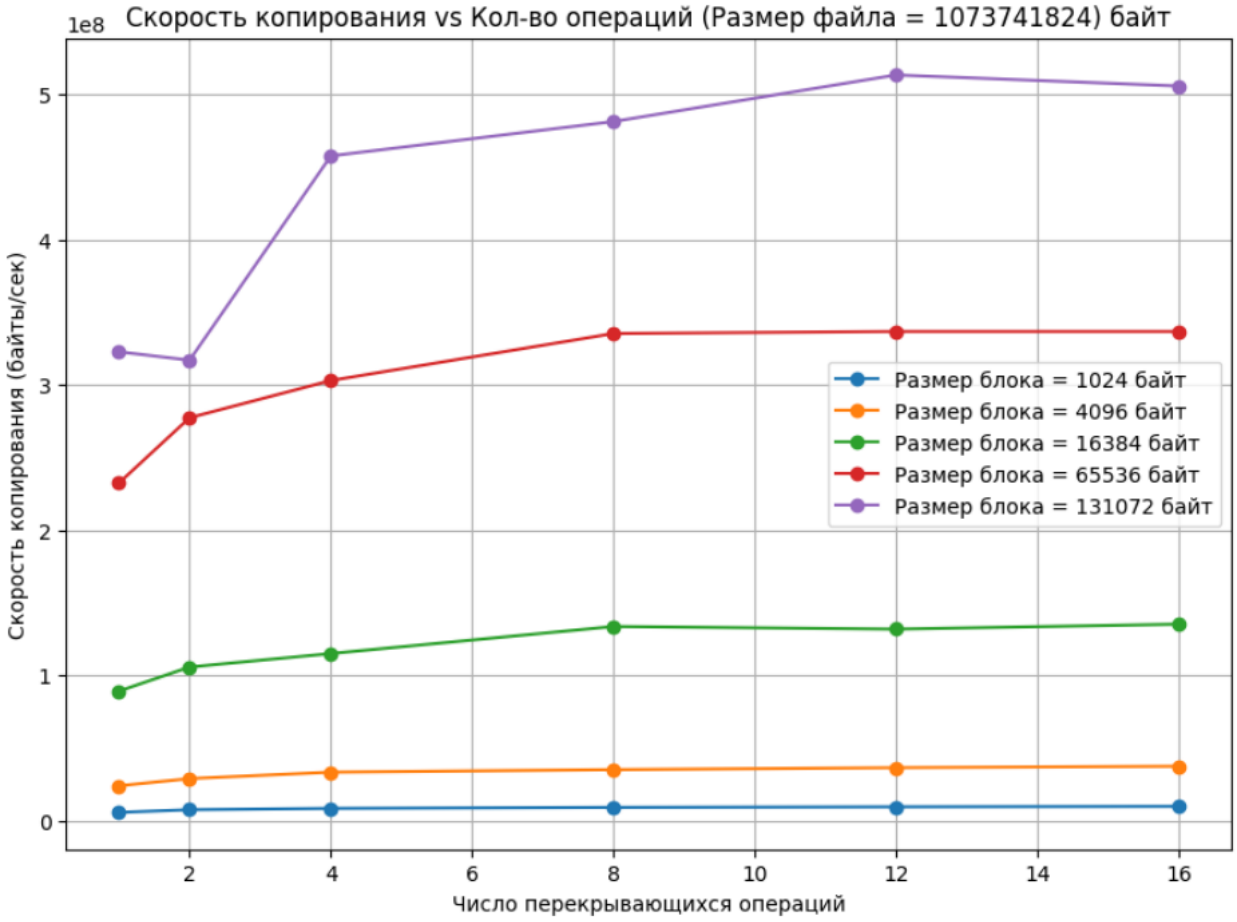
Графики зависимости числа перекрывающихся операций ввода и вывода, при котором достигается наибольшая скорость копирования файла:

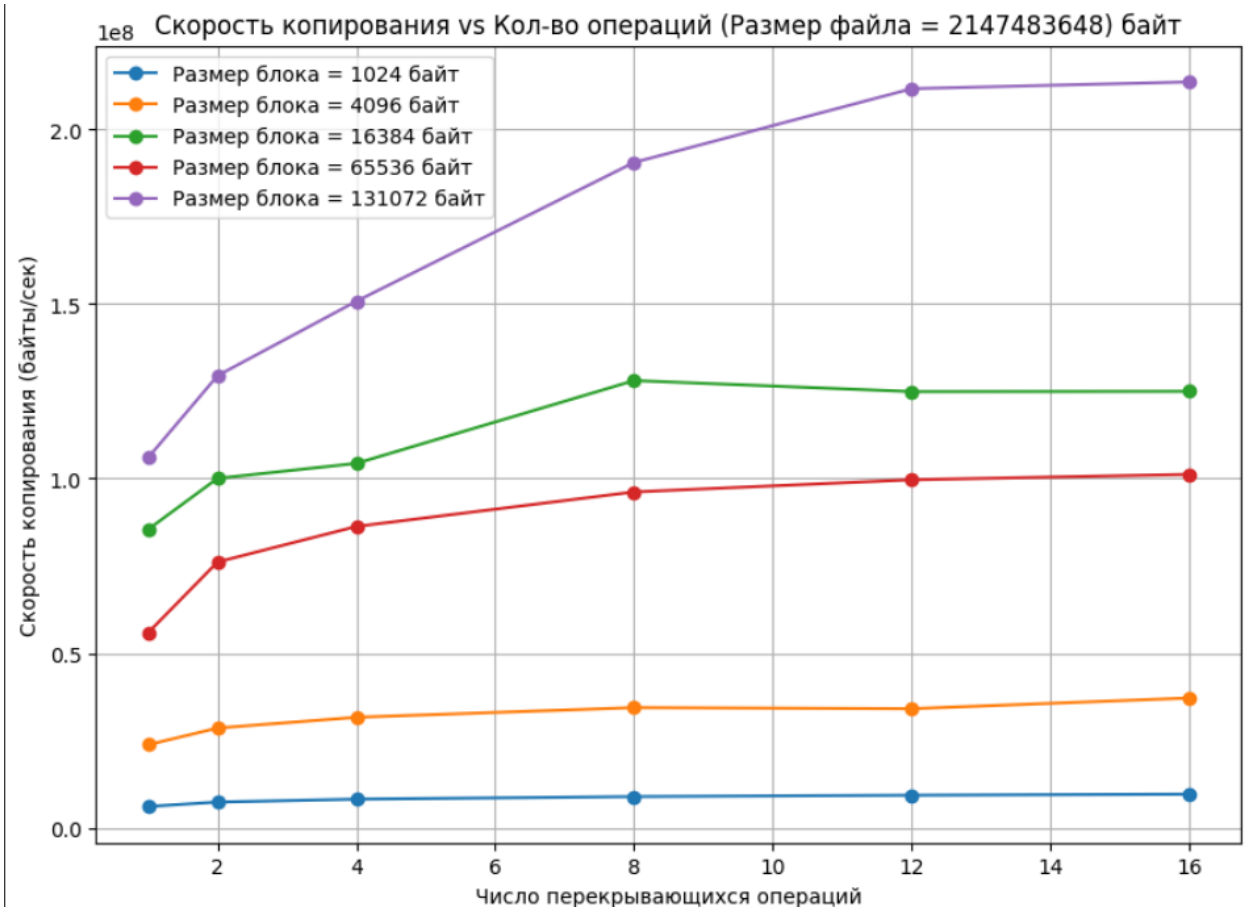

Скорость копирования vs Кол-во операций (Размер файла = 8388608) байт
1e8
Размер блока = 1024 байт
Размер блока = 4096 байт
Размер блока = 16384 байт
Размер блока = 65536 байт
Размер блока = 131072 байт
Скорость копирования (байты/сек)
Число перекрывающихся операций


Скорость копирования vs Кол-во операций (Размер файла = 16777216) байт
1e8
Размер блока = 1024 байт
Размер блока = 4096 байт
Размер блока = 16384 байт
Размер блока = 65536 байт
Размер блока = 131072 байт
Скорость копирования (байты/сек)
Число перекрывающихся операций


Скорость копирования vs Кол-во операций (Размер файла = 32048576) байт
1e8
Скорость копирования (байты/сек)
Число перекрывающихся операций
Размер блока = 1024 байт
Размер блока = 4096 байт
Размер блока = 16384 байт
Размер блока = 65536 байт
Размер блока = 131072 байт


Скорость копирования vs Кол-во операций (Размер файла = 134217728) байт
1e8
Скорость копирования (байты/сек)
Число перекрывающихся операций
Размер блока = 1024 байт
Размер блока = 4096 байт
Размер блока = 16384 байт
Размер блока = 65536 байт
Размер блока = 131072 байт


Скорость копирования vs Кол-во операций (Размер файла = 1073741824) байт
1e8
Скорость копирования (байты/сек)
Число перекрывающихся операций
Размер блока = 1024 байт
Размер блока = 4096 байт
Размер блока = 16384 байт
Размер блока = 65536 байт
Размер блока = 131072 байт
0
1
2
3
4
5
2
4
6
8
10
12
14
16



Из графиков зависимости скорости копирования от числа перекрывающихся операций видно, что увеличение количества операций сначала ускоряет копирование, но после некоторого значения скорость начинает снижаться или колебаться.

- Для небольших размеров блоков (1024, 4096 байт) оптимальное количество операций 4 или 8 – после этого производительность либо ухудшается, либо нестабильна.
- Для больших блоков (65536, 131072 байт) оптимальное значение операций – 12, так как после этого скорость копирования либо не увеличивается, либо даже ухудшается.

Таким образом, оптимальное число перекрывающихся операций – от 4 до 12, в зависимости от размера блока.

Анализ зависимости скорости копирования от размера блока данных показывает, что:

- При увеличении размера блока скорость копирования значительно возрастает.
- Например, для 8 МБ файла время копирования при блоке 1024 байта — 6.19 с, а при 131072 байтах — 0.02 с, что почти в 300 раз быстрее.

В ходе экспериментов, связанных с копированием файлов с использованием асинхронного ввода-вывода (AIO), были выявлены закономерности, позволяющие определить оптимальные параметры для наибольшей скорости копирования.

1. Влияние размера блока на скорость копирования

Анализ данных показал, что увеличение размера блока значительно ускоряет копирование файлов. Это связано с тем, что при передаче большего объёма данных за одну операцию уменьшается количество обращений к дисковой системе, снижая накладные расходы. Однако размер блока нельзя увеличивать бесконечно, так как при слишком больших значениях эффективность перестаёт расти из-за ограничений дисковой подсистемы и особенностей кэширования.

Оптимальным оказался размер блока 65536–131072 байт, при котором достигается максимальная скорость копирования без заметных потерь эффективности.

2. Оптимальное количество перекрывающихся операций

Параллельное выполнение нескольких операций ввода-вывода позволяет скрыть задержки, связанные с дисковой системой, но чрезмерное количество одновременных операций может привести к дополнительным накладным расходам.

Эксперимент показал, что наилучший результат достигается при 4–8 параллельных операциях, так как при этом удаётся эффективно загружать дисковую систему без перегрузки. При увеличении размера блока до 131072 байт оптимальное количество операций смещается в сторону 8, так как при больших блоках система способна лучше справляться с многопоточностью и пропускной способностью канала данных.

3. Эффективность асинхронного ввода-вывода (AIO)

Применение асинхронных операций позволило продемонстрировать, что грамотное управление потоками ввода-вывода значительно улучшает производительность. Использование пула асинхронных операций помогает уменьшить время простоя между запросами и задействовать механизм параллельного выполнения, что особенно важно при работе с медленными дисками или при многозадачной нагрузке.

При этом важную роль играет механизм ожидания завершения операций, такой как aio_suspend, который позволяет эффективно управлять процессом копирования, не создавая избыточной нагрузки на систему.

Вывод

Эксперимент подтвердил, что оптимальная комбинация параметров (размер блока и число параллельных операций) способна значительно ускорить процесс копирования файлов. На практике это знание можно использовать для оптимизации программ, работающих с большими объемами данных, а также для настройки системных параметров с целью повышения производительности ввода-вывода.

Текст программы

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <aio.h>
#include <errno.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>

struct aio_operation {
    struct aiocb aio;
    char *buffer;
    off_t original_offset;
    size_t block_size;  // real size of block
};

void wait_for_operation(struct aiocb *cb) {
    struct aiocb *aiolist[1] = {cb};
    while (aio_suspend(aiolist, 1, NULL) == -1 && errno == EINTR);
}

double async_copy(const char *src, const char *dst, size_t block_size, size_t
max_concurrent_ops) {
    struct timespec start_time, end_time;
    clock_gettime(CLOCK_MONOTONIC, &start_time);

    // open file descryptors
    int source_file_descryptor = open(src, O_RDONLY);
    if (source_file_descryptor < 0) {
        fprintf(stdout, "Error openning source file");
        close(source_file_descryptor);
        exit(1);
    }

    int destination_file_descryptor = open(dst, O_WRONLY | O_CREAT | O_TRUNC,
0666);
    if (destination_file_descryptor < 0) {
        fprintf(stdout, "Error openning destination file");
```

```
        close(source_file_descryptor);
        exit(1);
    }

    // get size file
    struct stat file_stat;
    fstat(source_file_descryptor, &file_stat);
    off_t file_size = file_stat.st_size;

    // get count of blocks
    const size_t total_blocks = (file_size + block_size - 1) / block_size;
    size_t blocks_processed = 0;

    while (blocks_processed < total_blocks) {
        struct aio_operation ops[max_concurrent_ops];

        // calculate batch_size - size of iteration block (count of io
operations)
        size_t batch_size;
        if ((total_blocks - blocks_processed) > max_concurrent_ops) batch_size =
max_concurrent_ops;
        else batch_size = total_blocks - blocks_processed;

        // start async read
        for (size_t i = 0; i < batch_size; i++) {
            const off_t current_offset = blocks_processed * block_size + i *
block_size;
            const size_t remaining = file_size - current_offset;
            const size_t op_size = (remaining > block_size) ? block_size :
remaining;

            // alocate size for buffer
            if (posix_memalign((void**)&ops[i].buffer, block_size, block_size)) {
                fprintf(stdout, "Memory allocation failed");
                exit(1);
            }

            ops[i].aio = (struct aiocb){
                .aio_fildes = source_file_descryptor,
                .aio_buf = ops[i].buffer,
                .aio_nbytes = op_size,
                .aio_offset = current_offset
            };
            ops[i].original_offset = current_offset;
            ops[i].block_size = op_size;

            if (aio_read(&ops[i].aio) < 0) {
                fprintf(stdout, "aio_read failed");
                exit(1);
            }
        }
```

```
        // wait operations
        for (size_t i = 0; i < batch_size; i++) {
            wait_for_operation(&ops[i].aio);

            ssize_t bytes_read = aio_return(&ops[i].aio);
            if (bytes_read < 0) {
                fprintf(stdout, "Read error at offset");
                exit(1);
            } else if ((size_t)bytes_read != ops[i].block_size) {
                fprintf(stdout, "Read size not equal to block size");
                exit(1);
            }

            // async write
            struct aiocb write_cb = {
                .aio_fildes = destination_file_descryptor,
                .aio_buf = ops[i].buffer,
                .aio_nbytes = (size_t)bytes_read,
                .aio_offset = ops[i].original_offset
            };

            if (aio_write(&write_cb) < 0) {
                fprintf(stdout, "aio_write failed");
                exit(1);
            }
            wait_for_operation(&write_cb);

            ssize_t bytes_written = aio_return(&write_cb);
            if (bytes_written < 0) {
                fprintf(stderr, "Write error at offset %ld: %s\n",
ops[i].original_offset, strerror(errno));
                exit(1);
            } else if ((size_t)bytes_written != (size_t)bytes_read) {
                fprintf(stderr, "Write size mismatch with block size");
                exit(1);
            }

            free(ops[i].buffer);
            blocks_processed++;
        }
    }

    clock_gettime(CLOCK_MONOTONIC, &end_time);
    double elapsed_time = (end_time.tv_sec - start_time.tv_sec) +
(end_time.tv_nsec - start_time.tv_nsec) / 1e9;
    // fprintf(stdout, "Copy completed successfully in %.6f seconds!\n",
elapsed_time);

    close(source_file_descryptor);
    close(destination_file_descryptor);
```

```
    return elapsed_time;
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <src> <dst> <block_size> <max_ops>\n",
argv[0]);
        return 1;
    }

    size_t block_size = atoi(argv[3]);
    size_t max_ops = atoi(argv[4]);

    double time = async_copy(argv[1], argv[2], block_size, max_ops);
    printf("Completed in %.6f seconds\n", time);

    return 0;
}

// clear file - `> filename.txt`
// get file size - `ls -lh filename.txt`
// create file with entered size - `head -c 10485760 < /dev/zero | tr '\0' '0' >
testfile.txt` - here it is 10 mbytes
```