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

ОТЧЕТ

по лабораторной работе № 1.2

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

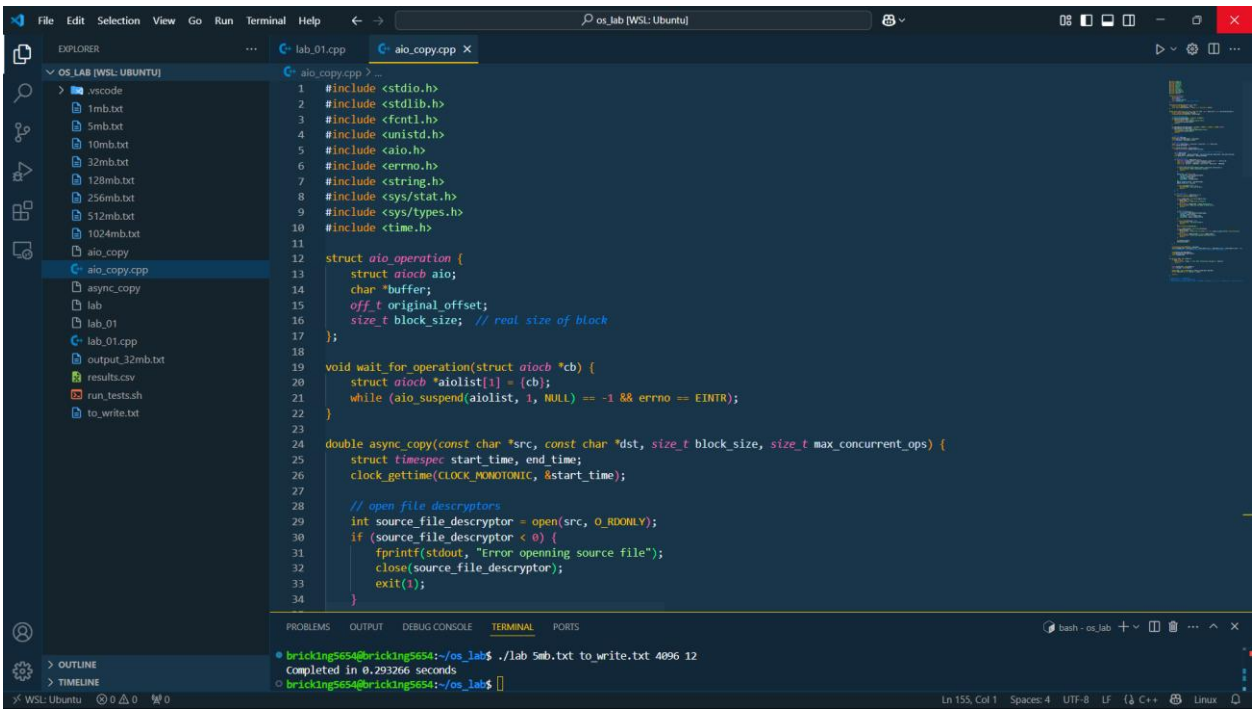
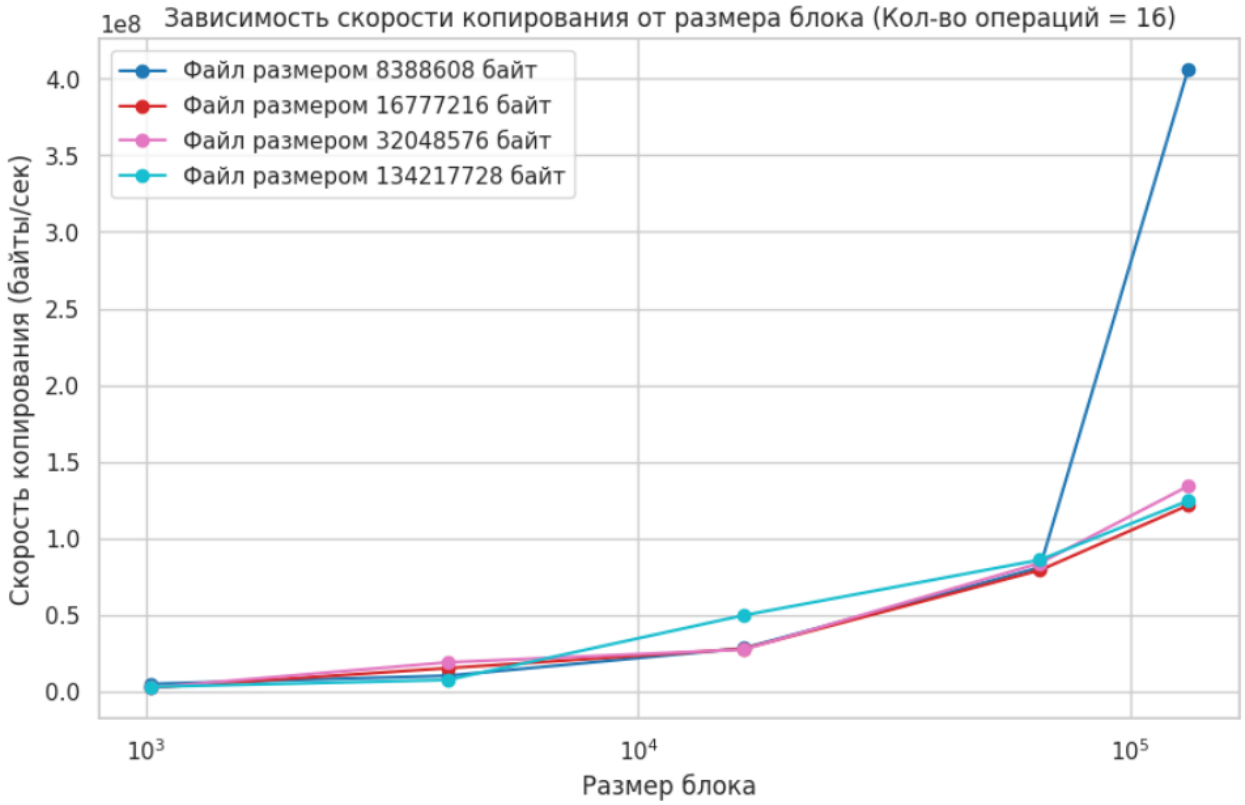
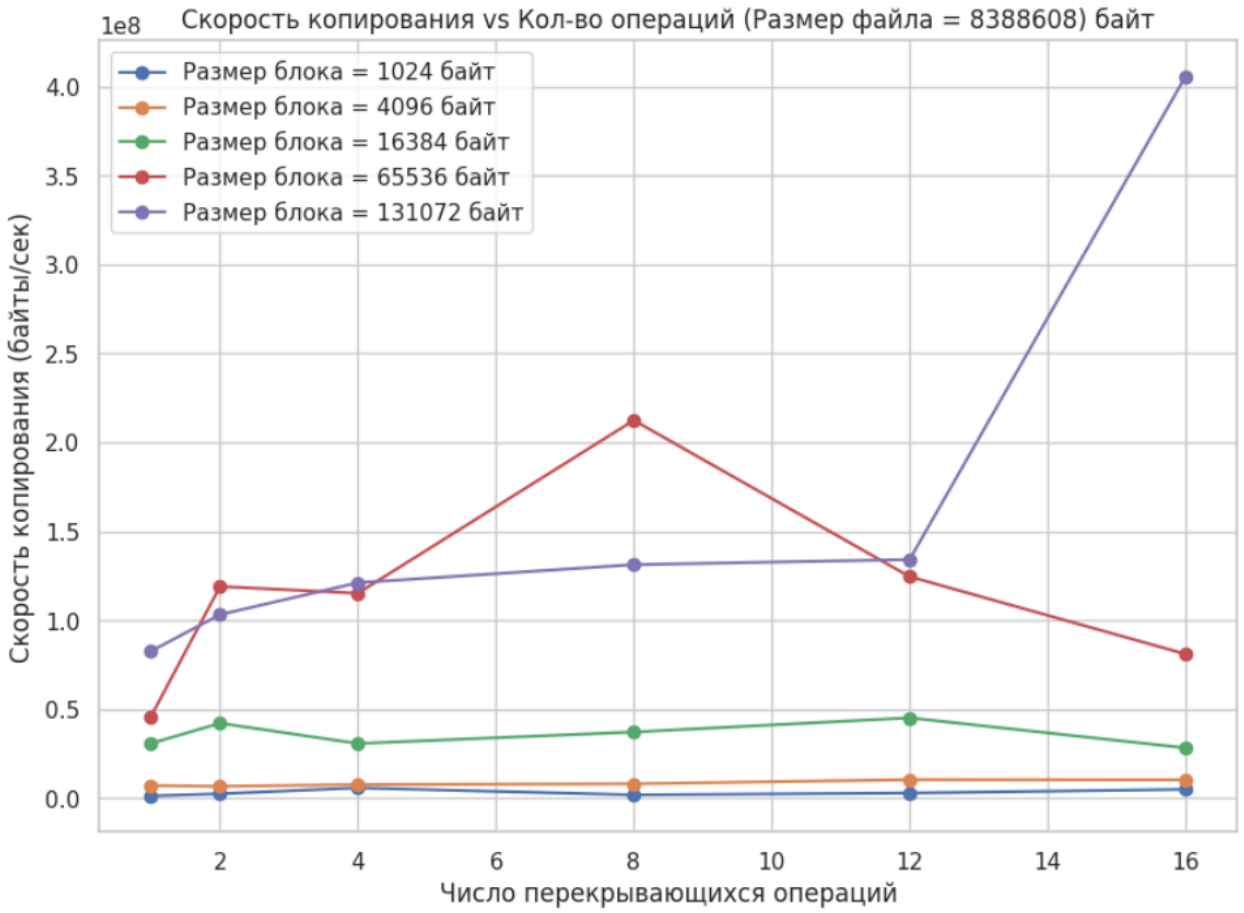
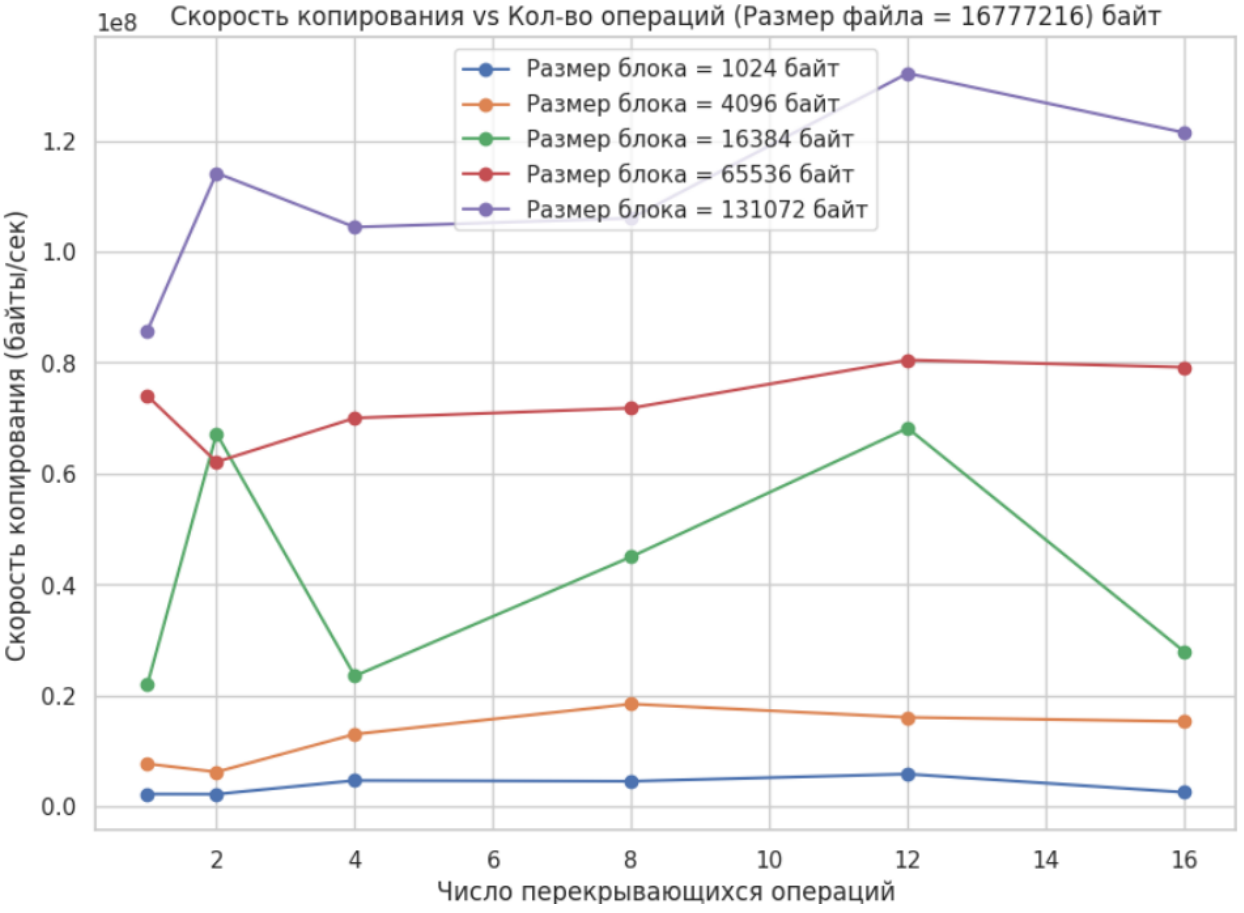


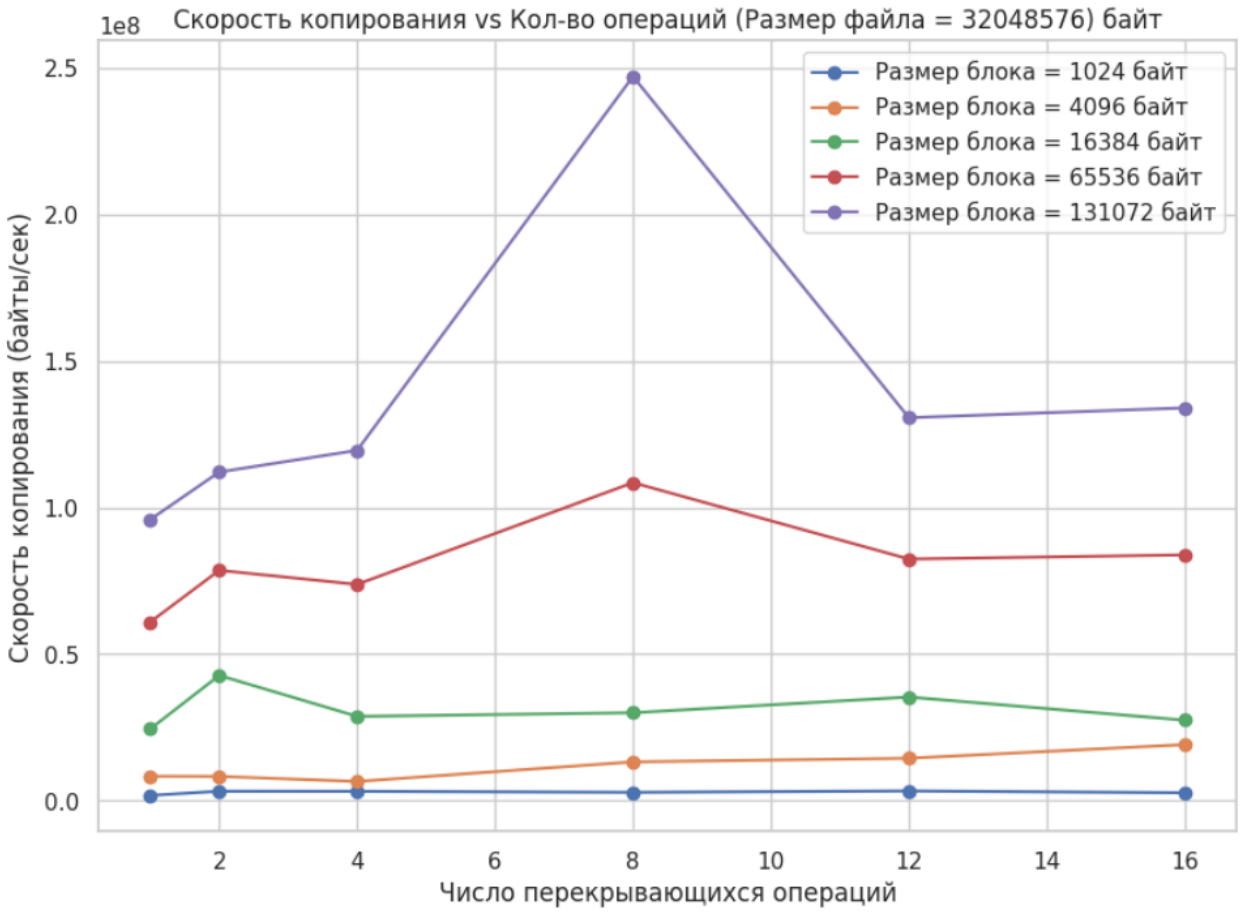
График зависимости скорости копирования от размера блока данных:

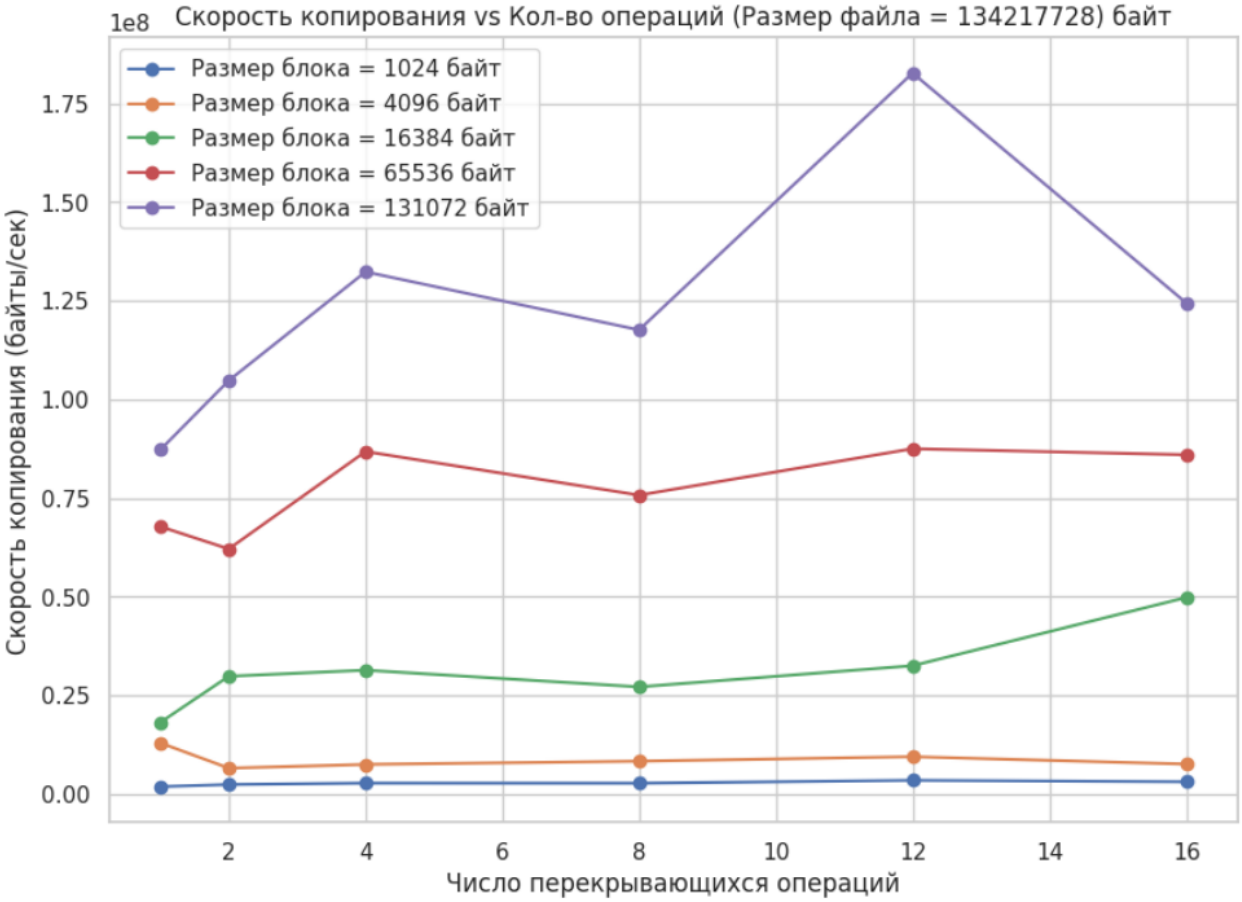


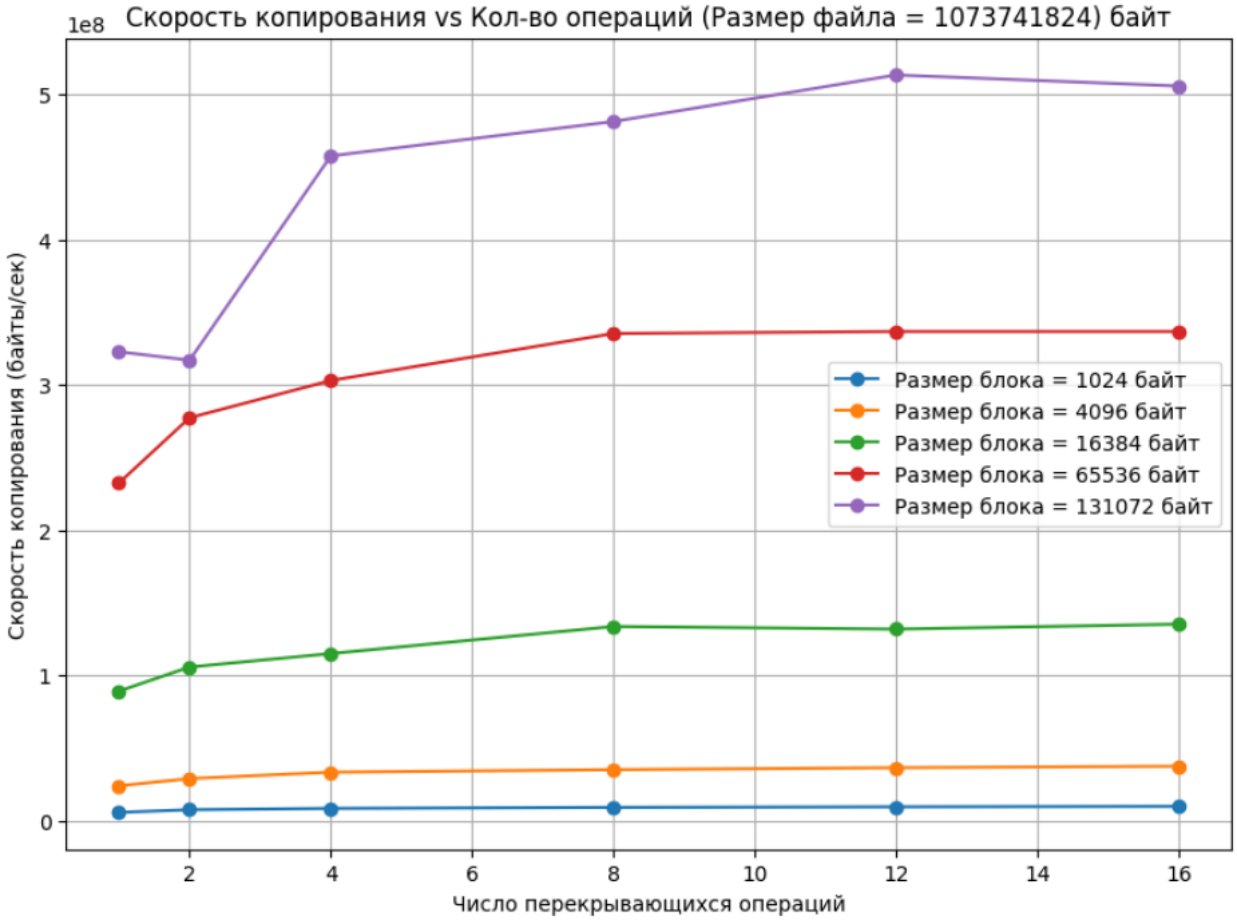
Графики зависимости числа перекрывающихся операций ввода и вывода, при котором достигается наибольшая скорость копирования файла:

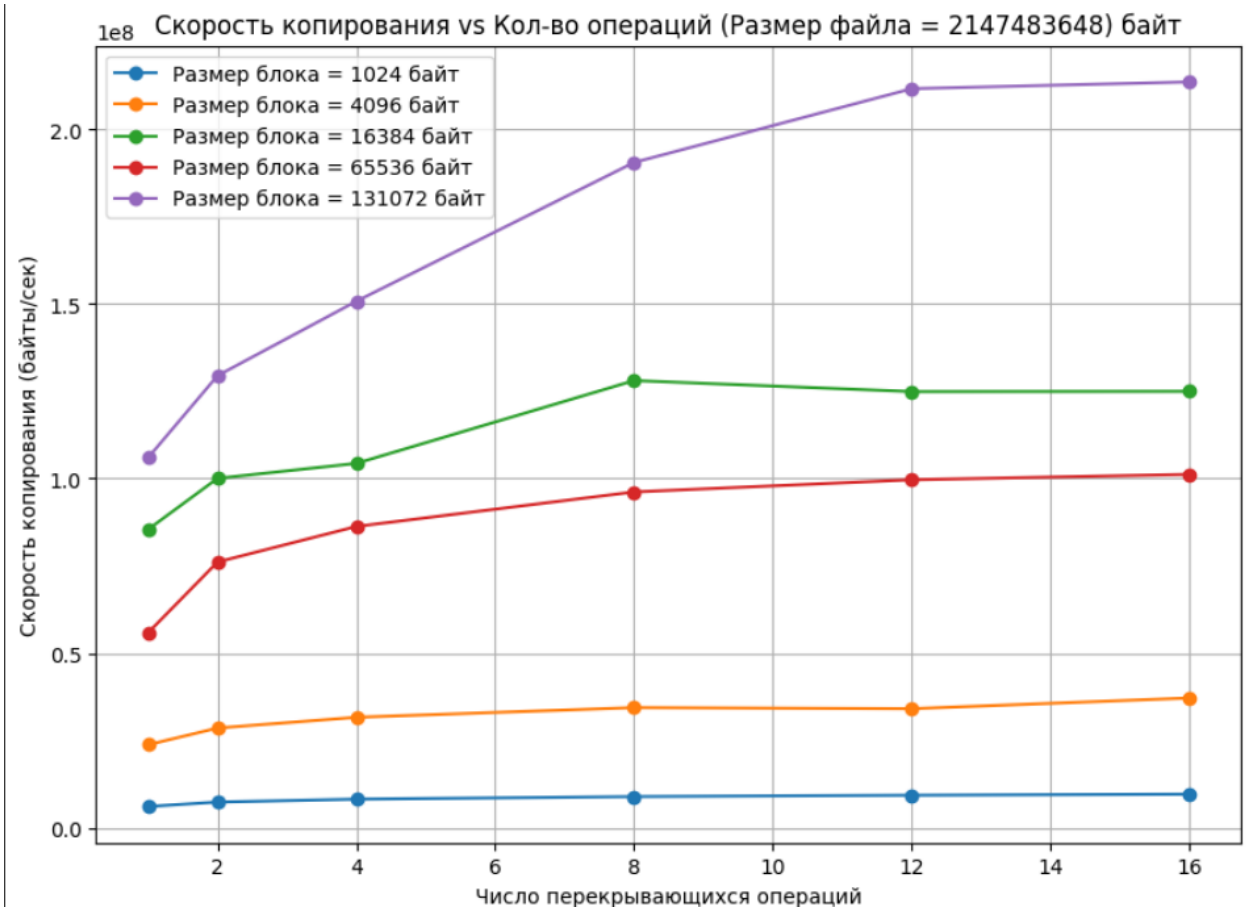

Скорость копирования vs Кол-во операций (Размер файла = 8388608) байт
1e8
Размер блока = 1024 байт
Размер блока = 4096 байт
Размер блока = 16384 байт
Размер блока = 65536 байт
Размер блока = 131072 байт
Скорость копирования (байты/сек)
Число перекрывающихся операций


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 данных, таких как 1024 или 4096 байт, оптимальное количество перекрывающихся операций ввода-вывода ограничено диапазоном от 4 до 8. Это связано с балансом между эффективностью параллельного выполнения и накладными расходами, которые возникают при управлении множеством асинхронных запросов. Когда размер блока невелик, каждая операция ввода-вывода обрабатывает относительно маленький объем данных. При этом система может эффективно распараллеливать выполнение нескольких таких операций, скрывая задержки, связанные с обращением к дисковой подсистеме. Увеличение числа перекрывающихся операций до 4 или 8 позволяет максимально загрузить дисковый контроллер и процессор, минимизируя время простоя между запросами.
  Однако дальнейшее увеличение числа операций (например, до 12 или 16) приводит к перегрузке системы. Каждая новая операция требует выделения ресурсов: памяти для буферов, процессорного времени для управления асинхронными вызовами и координации. При маленьких

блоках накладные расходы на управление начинают превышать выгоду от параллелизма.

- Для больших размеров блоков, таких как 65536 или 131072 байт, оптимальное число перекрывающихся операций смещается к 12. Здесь ключевую роль играет пропускная способность системы — способность дисковой подсистемы и каналов передачи данных обрабатывать большие объемы информации за единицу времени.
  Когда размер блока увеличивается, каждая операция ввода-вывода переносит значительно больше данных. Это снижает относительные накладные расходы на управление отдельными запросами, так как система тратит меньше времени на инициализацию и завершение операций по сравнению с полезной работой по передаче данных. Большие блоки позволяют глубже задействовать пропускную способность дисковой подсистемы. Однако превышение 12 операций (например, переход к 16 операциям) уже не дает прироста скорости, а иногда даже ухудшает результат. Причина кроется в ограничениях пропускной способности: дисковая система и шина данных имеют конечный предел скорости передачи. Когда операций становится слишком много, они начинают становиться в очереди, ожидая доступа к ресурсам, что увеличивает накладные расходы на управление и снижает общую эффективность

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 <aio.h>
#include <fcntl.h>
#include <signal.h>
#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

struct aio_operation {
    struct aiocb aio;
    char* buffer;
    off_t offset;
    bool is_write;
    int target_fd;
};

void aio_completion_handler(sigval_t sigval) {
    aio_operation* op = (aio_operation*)sigval.sival_ptr;
    ssize_t result = aio_return(&op->aio);

    if (result < 0) {
        fprintf(stderr, "%s error at offset %ld: %s\n",
                op->is_write ? "Write" : "Read",
                (long)op->offset, strerror(errno));
        free(op->buffer);
        delete op;
        return;
    }

    if (!op->is_write) {
        aio_operation* write_op = new aio_operation;
        *write_op = *op;
        write_op->is_write = true;
        write_op->aio = {};
        write_op->aio.aio_fildes = op->target_fd;
        write_op->aio.aio_buf = write_op->buffer;
        write_op->aio.aio_nbytes = result;
        write_op->aio.aio_offset = write_op->offset;
        write_op->aio.aio_sigevent.sigev_notify = SIGEV_THREAD;
        write_op->aio.aio_sigevent.sigev_notify_function =
aio_completion_handler;
        write_op->aio.aio_sigevent.sigev_value.sival_ptr = write_op;

        if (aio_write(&write_op->aio) < 0) {
            perror("aio_write failed");
            free(write_op->buffer);
            delete write_op;
```

```
        }
        delete op;
    } else {
        free(op->buffer);
        delete op;
    }
}

double copy_file_async(const char* src, const char* dst, size_t block_size, int
max_concurrent) {
    int src_fd = open(src, O_RDONLY);
    if (src_fd < 0) {
        perror("open src");
        exit(1);
    }

    int dst_fd = open(dst, O_WRONLY | O_CREAT | O_TRUNC, 0666);
    if (dst_fd < 0) {
        perror("open dst");
        close(src_fd);
        exit(1);
    }

    struct stat st;
    if (fstat(src_fd, &st) < 0) {
        perror("fstat");
        close(src_fd);
        close(dst_fd);
        exit(1);
    }

    off_t file_size = st.st_size;
    off_t offset = 0;

    clock_t start = clock();

    vector<aio_operation*> in_process;

    while (offset < file_size || !in_process.empty()) {
        while ((int)in_process.size() < max_concurrent && offset < file_size) {
            size_t size = min((size_t)(file_size - offset), block_size);
            aio_operation* op = new aio_operation;
            op->buffer = (char*)malloc(size);
            op->offset = offset;
            op->is_write = false;
            op->target_fd = dst_fd;

            memset(&op->aio, 0, sizeof(struct aiocb));
            op->aio.aio_fildes = src_fd;
            op->aio.aio_buf = op->buffer;
            op->aio.aio_nbytes = size;
```

```
            op->aio.aio_offset = offset;
            op->aio.aio_sigevent.sigev_notify = SIGEV_THREAD;
            op->aio.aio_sigevent.sigev_notify_function = aio_completion_handler;
            op->aio.aio_sigevent.sigev_value.sival_ptr = op;

            if (aio_read(&op->aio) < 0) {
                perror("aio_read");
                free(op->buffer);
                delete op;
                break;
            }
            in_process.push_back(op);
            offset += size;
        }
        usleep(5000);
        in_process.erase(remove_if(in_process.begin(), in_process.end(),
[](aio_operation* op) {
            return aio_error(&op->aio) != EINPROGRESS;
        }), in_process.end());
    }

    clock_t end = clock();
    close(src_fd);
    close(dst_fd);

    return (double)(end - start) / CLOCKS_PER_SEC;
}

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

    const char* src = argv[1];
    const char* dst = argv[2];
    size_t block_size = atoi(argv[3]);
    int max_ops = atoi(argv[4]);

    double duration = copy_file_async(src, dst, block_size, max_ops);
    printf("Copy completed in %.6f seconds.\n", duration);
    return 0;
}

// clear file - `> filename.txt`
// get file size - `ls -lh filename.txt`
// create file with entered size - `head -c 10485760
```