**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

**отчёт   
ЛАБОРАТОРНАЯ РАБОТА № 1**

**по дисциплине «Операционные Системы»**

**Тема: Управление файловой системой**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 |  | Сапронов К.Д. |
| Преподаватель |  | Тимофеев А.В. |

Санкт-Петербург

2025

**Цель работы:**

Исследование управления файловой системой с помощью Win32 API.

**Задание 1.1** Управление дисками, каталогами и файлами.

Создайте консольное приложение с меню (каждая выполняемая

функция и/или операция должна быть доступна по отдельному пункту меню), которое выполняет:

- вывод списка дисков

- вывод информации (тип, том, объем) для одного из выбранных дисков

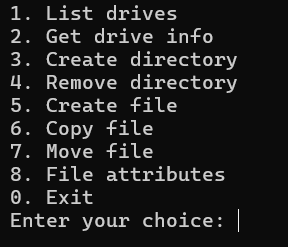
- создание и удаление каталогов

- работу с файлами (создание, удаление, перемещение и перемещение с параметрами)  
- проверка и изменение атрибутов и времени файла

Запустить приложение и проверить его работоспособность на нескольких выборах вводимых данных. Запротоколировать результаты в отчёт. Дать комментарии относительно выполнения функций Win32 API.

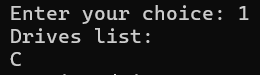
**Работа программы:**

При запуске программы пользователь видит главное меню. В нем может выбрать одну из функций программы (опции 1-8) или выйти из нее (опция 0).



**1. Список дисков:**

При выборе этого пункта программа отображает список доступных пользователю дисков. Для этого вызывается функция GetLogicalDrives(), которая возвращает битовую маску. В этой маске каждый бит, начиная справа, соответствует букве диска (A — первый бит, B — второй и т.д.). Программа обращается к системным структурам ОС, где хранятся данные о подключённых логических дисках. Если бит установлен, то программа выводит соответствующую букву.



**2. Получение информации о диске:**

В этом пункте используются три функции для получения полной информации о выбранном пользователем диске:

* **GetDriveType()** — определяет тип диска (например, съёмный, жёсткий, CD/DVD, сетевой и т.д.). Windows использует драйверы и ядро, чтобы определить, какое устройство соответствует указанной букве диска.
* **GetVolumeInformation()** — возвращает метаданные тома, включая:
* Имя тома (Volume Name).
* Тип файловой системы (например, NTFS, FAT32).
* Серийный номер тома.
* Максимальную длину имени файла.
* Набор возможностей файловой системы.

Для NTFS (наиболее распространённой файловой системы) проверяются следующие флаги:

* FILE\_CASE\_SENSITIVE\_SEARCH: Чувствительность к регистру в именах файлов.
* FILE\_CASE\_PRESERVED\_NAMES: Сохранение регистра в именах файлов.
* FILE\_UNICODE\_ON\_DISK: Поддержка Unicode в именах файлов.
* FILE\_PERSISTENT\_ACLS: Управление доступом на основе списков ACL.
* FILE\_FILE\_COMPRESSION: Поддержка сжатия файлов.
* FILE\_VOLUME\_IS\_COMPRESSED: Сжатие тома.
* FILE\_SUPPORTS\_ENCRYPTION: Поддержка шифрования файлов (EFS).
* FILE\_SUPPORTS\_SPARSE\_FILES: Поддержка разреженных файлов.
* FILE\_SUPPORTS\_REPARSE\_POINTS: Поддержка точек повторного анализа (символические ссылки, точки подключения).
* FILE\_SUPPORTS\_HARD\_LINKS: Поддержка жёстких ссылок.
* FILE\_SUPPORTS\_EXTENDED\_ATTRIBUTES: Поддержка расширенных атрибутов.
* FILE\_SUPPORTS\_TRANSACTIONS: Поддержка транзакций на уровне файловой системы.
* **GetDiskFreeSpace()** — возвращает информацию о свободном и занятом пространстве на диске:
* Общий объём тома.
* Объём свободного места.
* Объём доступного места для текущего пользователя.

Эта функция запрашивает данные у драйвера файловой системы, чтобы получить актуальную информацию о занятости диска.

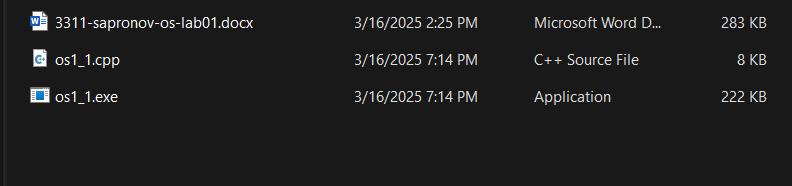


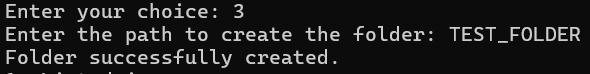
В данном случае тип диска равен 3, что соответствует DRIVE\_FIXED – жесткому диску.

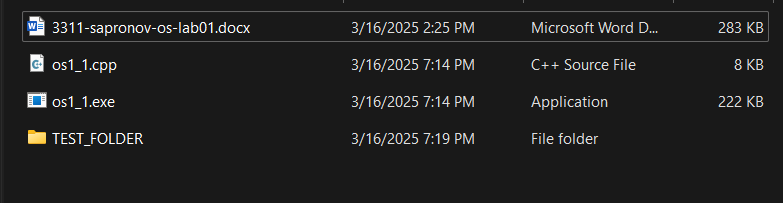
**3. Создание каталога:**

При выборе 3 вызывается функция **CreateDirectory()**. На этом этапе операционная система пытается зарезервировать имя папки в файловой системе и добавить соответствующую запись в таблицу каталогов (например, NTFS или FAT). Windows также проверяет, есть ли у пользователя необходимые права для создания папки в указанном месте (например, права записи в родительской директории и настройки ACL). Если все проверки пройдены, создаётся новая запись в каталоге, а на диске может быть выделен дополнительный блок метаданных для новой папки.

Изначальное содержание папки:

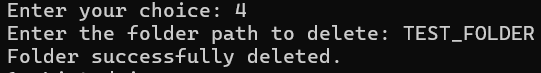


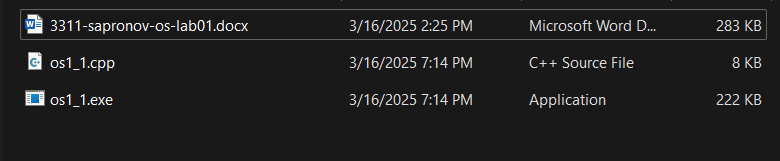




**4. Удаление каталога:**

При выборе этого пункта вызывается функция **RemoveDirectory()**: Windows проверяет, пуста ли данная директория (**RemoveDirectory()** не удалит папку, если в ней есть файлы или подпапки). ОС убеждается, что у программы есть нужные права доступа (запись, удаление) и что объект не занят другим процессом.





5. Создание файла:

Создание файла осуществляется с помощью функции WinAPI **CreateFile()**, которая возвращает дескриптор типа **HANDLE**. Дескриптор — это специальное значение, которое операционная система выдаёт процессу, подтверждая право на взаимодействие с конкретным файлом. Он содержит информацию о:

* Связанном файле или ресурсе.
* Режиме доступа (чтение, запись или оба).
* Процессе, которому он принадлежит.
* И других параметрах.

После завершения работы с файлом необходимо закрыть дескриптор с помощью функции **CloseHandle()**, чтобы освободить ресурсы системы (например, снять блокировки и освободить память, связанную с дескриптором).

Функция **CreateFile()** принимает несколько параметров:

1. Путь к файлу — указывает, где и под каким именем будет создан файл.
2. Права доступа:

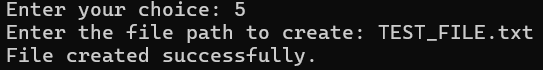
* GENERIC\_READ — разрешает чтение.
* GENERIC\_WRITE — разрешает запись.
* GENERIC\_READ | GENERIC\_WRITE — разрешает и чтение, и запись.

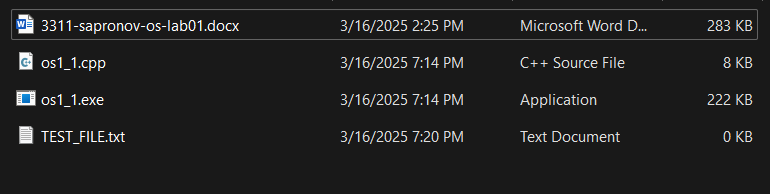
1. Режим совместного доступа (sharing mode) — параметр, определяющий, могут ли другие процессы одновременно работать с этим файлом. Значение 0 означает, что доступ для других процессов запрещён.
2. Настройки безопасности (Security Attributes) — если передать NULL, используются настройки по умолчанию (без специального наследования или ACL).
3. Режим создания: в программе используется параметр CREATE\_NEW — создаёт файл, но возвращает ошибку, если файл уже существует.
4. Атрибуты файла — например, FILE\_ATTRIBUTE\_NORMAL задаёт базовые атрибуты (файл не скрытый, не системный и т.д.). Атрибуты можно изменить позже с помощью отдельной функции.
5. Шаблон— если передать NULL, шаблон не используется.

При успешном создании файла программа выводит сообщение: «File created successfully» и закрывает дескриптор с помощью CloseHandle(hFile).

На уровне операционной системы:

* Создаётся новая запись о файле в файловой системе (например, в MFT для NTFS).
* Новому файлу назначается атрибут ARCHIVE.
* Кластеры на диске выделяются при первой записи данных в файл.





**6. Копирование файла:**

В пункте 6 выполняется копирование указанного файла в новое место по заданному пути. Для этого используется функция **CopyFile()**, которая принимает три параметра:

1. Путь к исходному файлу, который нужно скопировать.
2. Путь, куда файл будет скопирован.
3. Флаг перезаписи:

* FALSE — если файл уже существует, он будет перезаписан.
* TRUE — если файл уже существует, функция вернёт ошибку.

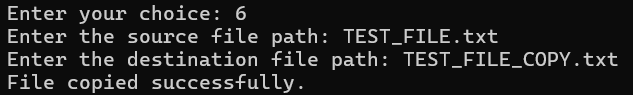
Функция **CopyFile()** работает только с файлами и не поддерживает копирование папок. Она ожидает, что **первый параметр** указывает на существующий файл, а **второй** — на имя нового или перезаписываемого файла.

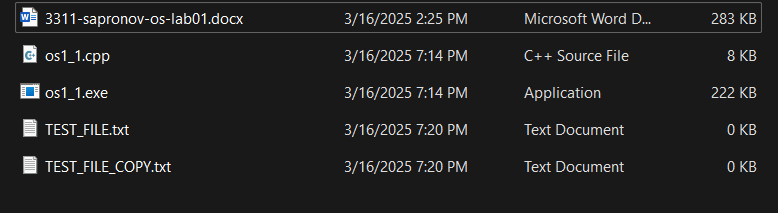
Функция завершится ошибкой, если первый параметр указывает на несуществующий файл, на папку, или если у пользователя недостаточно прав для чтения файла или записи в указанное место.

На уровне операционной системы:

1. Открывается исходный файл для чтения.
2. Создаётся (или открывается) целевой файл в режиме записи.
3. Содержимое файла копируется блоками. При этом могут использоваться механизмы буферизации и кэширования.
4. Если файл большой, Windows автоматически разбивает процесс копирования на части.
5. По завершении копирования дескрипторы исходного и целевого файлов закрываются внутри функции **CopyFile()**.

При успешном выполнении программа выводит сообщение: «File copied successfully».





**7. Перемещение файла**

В процессе перемещения или переименования файла операционная система выполняет несколько проверок:

1. Существует ли исходный файл.
2. Есть ли у вызывающего процесса достаточно прав для выполнения операции.

Если адреса находятся на одном логическом томе, Windows просто обновляет внутренние записи в файловой системе, указывая, что файл теперь имеет новое имя или расположение. Это происходит почти мгновенно, так как не требует физического копирования данных.

Если же они находятся на разных томах (например, C: → D:), Windows обычно выполняет «двойную операцию»:

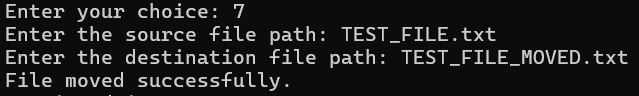
1. Копирует данные на новый том.
2. Удаляет исходный файл.

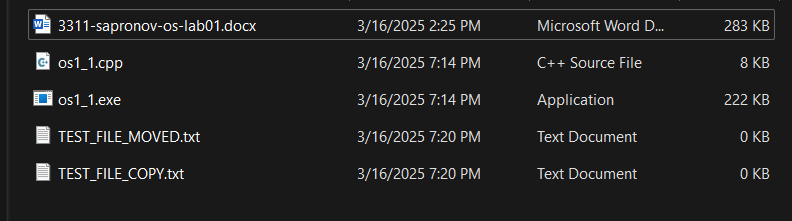
Однако стандартная функция MoveFile() не всегда поддерживает такие сценарии. Для более сложных случаев используется MoveFileEx() с дополнительными флагами. В некоторых ситуациях при попытке перемещения между томами MoveFile() может вернуть ошибку.

Параметр flags представляет собой комбинацию флагов (побитовое «или»), таких как:

* MOVEFILE\_REPLACE\_EXISTING — если файл по пути destination уже существует, он будет заменён.
* MOVEFILE\_COPY\_ALLOWED — разрешает копирование при перемещении между разными томами. Вместо мгновенного переименования файл копируется на новый том, а исходный удаляется.
* MOVEFILE\_DELAY\_UNTIL\_REBOOT — откладывает перемещение или удаление до перезагрузки системы. Это полезно, например, для замены системных файлов, которые заняты процессами.

Эти флаги позволяют гибко управлять процессом перемещения файлов в зависимости от требований и условий.





1. **Проверка и изменение атрибутов файла:**

В пункте 8 программа вызывает функцию GetFileAttributes(), чтобы получить текущие атрибуты файла, такие как READONLY, HIDDEN, SYSTEM, ARCHIVE и другие. Если функция возвращает INVALID\_FILE\_ATTRIBUTES, это означает, что файл не существует или произошла ошибка.

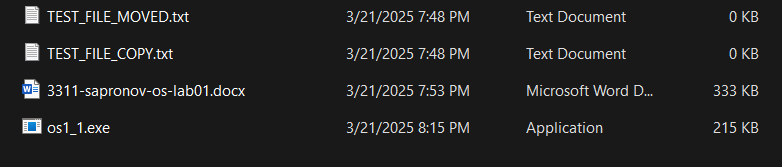
Основные атрибуты файла:

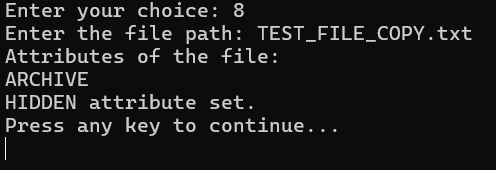
* FILE\_ATTRIBUTE\_READONLY - Только чтение файл.
* FILE\_ATTRIBUTE\_HIDDEN - Скрытый файл.
* FILE\_ATTRIBUTE\_SYSTEM - Системный файл.
* FILE\_ATTRIBUTE\_DIRECTORY - Каталог.
* FILE\_ATTRIBUTE\_ARCHIVE - Архивный файл.
* FILE\_ATTRIBUTE\_ENCRYPTED - Зашифрованный файл.
* FILE\_ATTRIBUTE\_COMPRESSED - Сжатый файл.
* FILE\_ATTRIBUTE\_OFFLINE - Оффлайн файл.
* FILE\_ATTRIBUTE\_TEMPORARY - Временный файл.

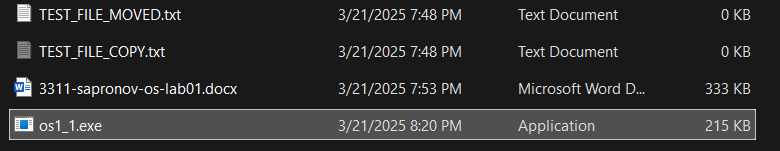
Программа позволяет пользователю изменить атрибуты файла отчета (этого документа) с помощью функции SetFileAttributes(). Эта функция устанавливает или снимает определённые флаги в метаданных файла. Например:

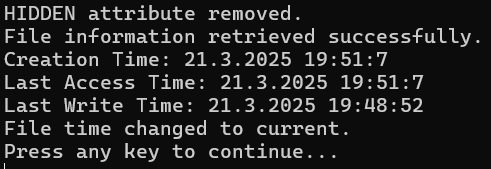
* Если снять флаг READONLY, файл можно будет изменять или удалять без дополнительных запросов.
* Если установить флаг HIDDEN, Проводник Windows (при стандартных настройках) перестанет отображать этот файл.

Таким образом, пользователь может гибко управлять атрибутами файлов, изменяя их поведение и видимость в системе.











**Выводы по выполнению задания**

В ходе выполнения лабораторной работы было разработано консольное приложение, демонстрирующее использование функций Win32 API для управления файловой системой. Программа реализует основные операции с дисками, каталогами и файлами, такие как вывод списка дисков, получение информации о дисках, создание и удаление каталогов, работа с файлами (создание, копирование, перемещение), а также проверка и изменение атрибутов и временных меток файлов.

Программа успешно использует функцию GetLogicalDrives() для получения списка доступных дисков, что демонстрирует, как Win32 API позволяет получить информацию о подключённых логических дисках. Для получения подробной информации о диске применяются функции GetDriveType(), GetVolumeInformation() и GetDiskFreeSpaceEx(). Эти функции показывают, как Win32 API предоставляет доступ к данным о типе диска, файловой системе, объёме свободного и занятого пространства, а также другим метаданным тома.

Для работы с каталогами программа использует функции CreateDirectory() и RemoveDirectory(). Это демонстрирует, как Win32 API позволяет управлять структурой файловой системы, создавая и удаляя папки. При этом учитываются права доступа пользователя и состояние каталога (например, удаление возможно только для пустых папок).

Создание файлов осуществляется с помощью функции CreateFile(), которая возвращает дескриптор файла. Это показывает, как Win32 API позволяет взаимодействовать с файлами на низком уровне, управляя их атрибутами и режимами доступа. Функция CopyFile() используется для копирования файлов, а MoveFile() и MoveFileEx() — для их перемещения или переименования. Эти функции демонстрируют, как Win32 API обрабатывает операции с файлами, включая копирование между томами и работу с флагами (например, MOVEFILE\_REPLACE\_EXISTING).

Для управления атрибутами и временем файлов программа использует функции GetFileAttributes() и SetFileAttributes(), а также GetFileInformationByHandle(), FileTimeToSystemTime() и SetFileTime(). Это показывает, как Win32 API позволяет управлять видимостью, доступностью и временными характеристиками файлов.

Важным аспектом работы с файловой системой через Win32 API является использование дескрипторов (HANDLE). При открытии или создании файла операционная система выдаёт программе дескриптор, содержащий информацию о режиме доступа и идентификаторе файла. После завершения работы с файлом дескриптор должен быть закрыт с помощью CloseHandle(), чтобы освободить ресурсы и не блокировать файл для других процессов. Это ключевой механизм, обеспечивающий безопасность и корректное управление ресурсами.

Кроме того, программа демонстрирует, как файловая система управляет кластерами (блоками хранения). При создании файлов система резервирует кластеры, а при удалении — помечает их как свободные. Это позволяет эффективно управлять дисковым пространством, даже если физическая запись данных происходит только при фактическом заполнении файла.

Задание 1.1 демонстрирует, как на низком уровне (через Win32 API) можно полностью управлять объектами файловой системы: от создания каталога и получения списка дисков до точной настройки атрибутов и времени файлов. Это подчеркивает гибкость Win32 API для работы с файловой системой, а также важность понимания её внутренних механизмов.

**Задание 1.2.** Копирование файла с помощью операций перекрывающегося ввода-вывода в Linux.

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

1. Создать консольное приложение, которое выполняет открытие/создание файлов и измерение продолжительности выполнения операции копирования файла

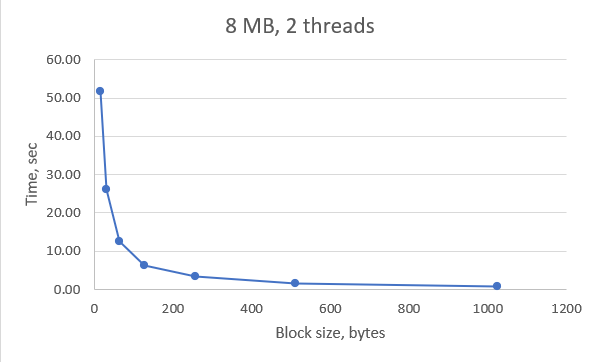
2. Запустите приложение и проверьте его работоспособность на копировании файлов разного размера для ситуации с перекрывающимся выполнением одной операции ввода и одной операции вывода. Выполните эксперимент для разного размера копируемых блоков, постройте график зависимости скорости копирования от размера блока данных.

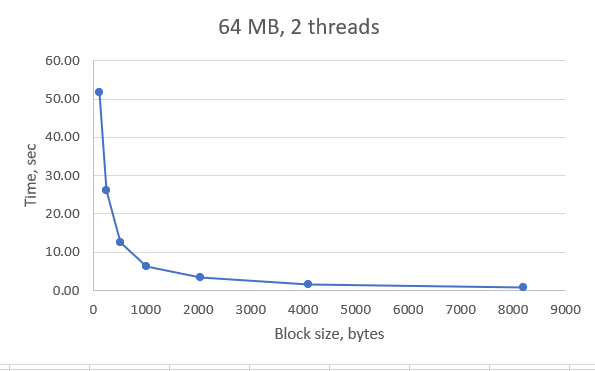
Определите оптимальный размер блока данных, при котором скорость копирования наибольшая. Запротоколируйте результаты в отчет.

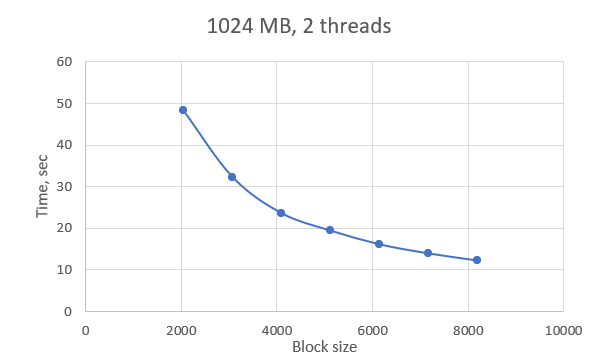
3. Произведите замеры времени выполнения приложения для разного числа перекрывающихся операций ввода и вывода (1, 2, 4, 8, 12, 16).

По результатам измерений постройте график зависимости и определите число перекрывающихся операций ввода и вывода, при котором достигается наибольшая скорость копирования файла. Запротоколируйте результаты в отчет.

* + 1. **Графики для одной операции ввода и одной – вывода.**

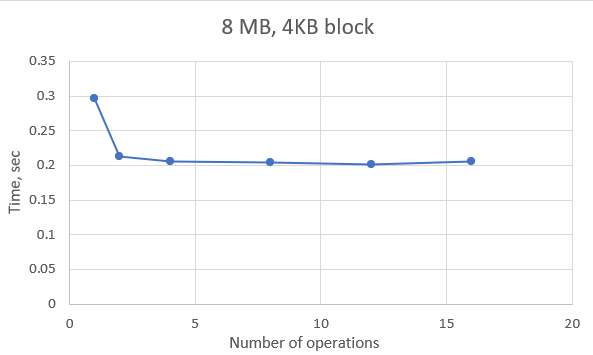


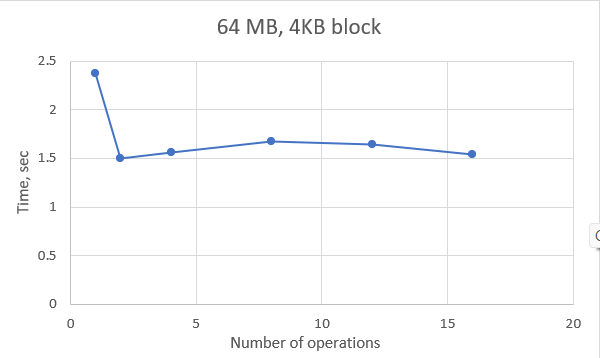


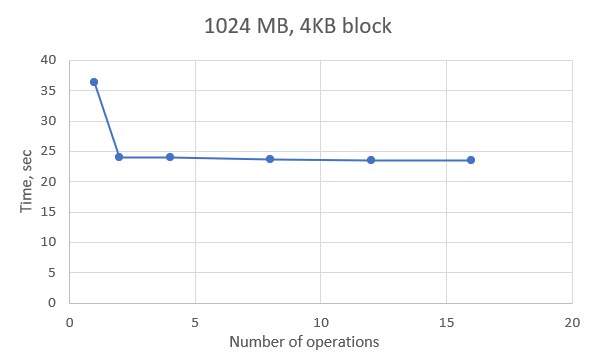


По графикам видно, что наиболее оптимальным значением является 4 КБ, т.к. он является оптимальным размером для больших объемов данных (1 ГБ). Увеличение размера блока приводит к незначительному повышению скорости, а его уменьшение приводит к заметному падению. Для меньших размеров, например, для 64 МБ, ситуация схожая – дальнейшее увеличение приводит к незначительному росту скорости, а уменьшение блока, хоть и не приводит к значительному падению скорости, уже невыгодно для больших лбъемов данных.

* + 1. **Графики для разного количества операций.**







По графикам видим, что 2 операции (т.е. по одной ввода и вывода) является наиболее оптимальным решением, т.к. большее количество операций дает небольшую прибавку к скорости (если дает вообще), при этом задействуется кратно большее число ресурсов. Уменьшение до 1 операции, очевидно, сильно ухудшает скорость копирования из-за того, что ей приходится поочередно выполнять чтение и запись.

**Выводы:**

Наиболее эффективным размером блока данных оказался размер 4 КБ. Он демонстрируют стабильно высокую скорость копирования как для малых, так и для больших файлов.

При увеличении размера блока данных свыше 4 КБ наблюдалось меньшее повышение скорости копирования. Это связано с тем, что большие блоки данных уменьшают количество системных вызовов, что снижает накладные расходы на переключение контекста. Однако дальнейшее увеличение размера блока не приводит к существенному приросту производительности, так как ограничивающим фактором становится пропускная способность диска или файловой системы.

При уменьшении размера блока ниже 4 КБ скорость копирования заметно падает. Это связано с увеличением количества системных вызовов и накладных расходов на обработку каждого запроса, что приводит к снижению общей производительности.

Наиболее эффективным оказалось использование двух перекрывающихся операций (одна операция ввода и одна операция вывода). Это позволяет достичь баланса между параллелизмом и накладными расходами на управление несколькими операциями.

При увеличении количества операций свыше двух (до 4, 8, 12 или 16) наблюдалось незначительное увеличение скорости копирования или его отсутствие. Это связано с тем, что дополнительные операции начинают конкурировать за ресурсы системы (например, за доступ к диску или кэшу), что приводит к увеличению накладных расходов и снижению эффективности.

Использование только одной операции ввода-вывода приводит к снижению скорости копирования, так как отсутствует параллелизм между операциями чтения и записи.