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

ОТЧЁТ

по лабораторной работе №1

1. по дисциплине «Операционные системы»
2. Тема: Управление файловой системой

|  |  |  |
| --- | --- | --- |
| Студент гр. 3312 |  | Нкусси ф.М |
| Преподаватель |  | Тимофеев А.В. |

**Оглавление**

[1. Введение](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214820) 3

[2. Управление дисками, каталогами и файлами 5](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214821)

[2.1.](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214822) **Вывод списка дисков** 5

[2.2. Вывод информации о диске и размер свободного пространства 5](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214823)

[2.3. Создание и удаление заданных каталогов 5](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214824)

[2.4. Создание файлов в новых каталогах 6](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214825)

[2.5. Копирование и перемещение файлов между каталогами 7](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214826)

[2.6. Анализ и изменение атрибутов файлов 9](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214827)

[2.7. Исходный код программы 24](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214828)

[2.8. Выводы 27](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214829)

[3. Копирование файла с помощью перекрывающих операций ввода-вывода 28](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214830)

[3.1. Создание и запуск консольного приложения 28](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214831)

[3.2. Проверка приложения на разных размерах копируемых блоков 29](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214832)

[3.3. Проверка приложения на разном числе операция ввода-вывода 30](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214833)

[3.4. Исходный код программы 31](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214834)

[3.5. Выводы 36](../../../../C:/Users/RedMo/Downloads/9308_SobolevMS_OS_LW1.docx" \l "_Toc84214835)

1. **1. Введение**

Тема работы: Управление файловой системой.

Цель работы: Исследование управления файловой системой с помощью Win32 API.

**Указания к выполнению**

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

1. Создайте консольное приложение с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которое выполняет:

- вывод списка дисков (функции Win32 API – GetLogicalDrives, GetLogicalDriveStrings);

- для одного из выбранных дисков вывод информации о диске и размер свободного пространства (функции Win32 API – GetDriveType, GetVolumeInformation, GetDiskFreeSpace);

- создание и удаление заданных каталогов (функции Win32 API – CreateDirectory, RemoveDirectory);

- создание файлов в новых каталогах (функция Win32 API – CreateFile);

- копирование и перемещение файлов между каталогами с возможностью выявления попытки работы с файлами, имеющими совпадающие имена (функции Win32 API – CopyFile, MoveFile, MoveFileEx);

- анализ и изменение атрибутов файлов (функции Win32 API – GetFileAttributes, SetFileAttributes, GetFileInformationByHandle, GetFileTime, SetFileTime).

2. Запустите приложение и проверьте его работоспособность на нескольких наборах вводимых данных. Запротоколируйте результаты в отчёт. Дайте свои комментарии в отчёте относительно выполнения функций Win32 API.

3. Подготовьте итоговый отчёт с развернутыми выводами по заданию.

**Задание 1.2.** Копирование файла с помощью операций

перекрывающегося ввода-вывода (будет изменено на выполнение в Linux).

Приложение должно копировать существующий файл в новый файл,

«одновременно» выполняя n перекрывающихся операций ввода-вывода

(механизм APC) блоками данных кратными размеру кластера.

**Указания к выполнению.**

1. Создайте консольное приложение, которое выполняет: −

открытие/создание файлов (функция Win32 API – CreateFile,

обязательно использовать флаги FILE\_FLAG\_NO\_BUFFERING и

FILE\_FLAG\_OVERLAPPED);

∙ файловый ввод-вывод (функции Win32 API – ReadFileEx,

WriteFileEx) блоками кратными размеру кластера;

∙ ожидание срабатывания вызова функции завершения (функция Win32

API – SleepEx);

∙ измерение продолжительности выполнения операции копирования файла

(функция Win32 API – TimeGetTime).

2. Запустите приложение и проверьте его работоспособность на копировании файлов разного размера для ситуации с перекрывающимся выполнением одной операции ввода и одной операции вывода (для сравнения файлов используйте консольную команду FC). 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. Дайте свои комментарии в отчете относительно выполнения функций Win32 API.

3. Произведите замеры времени выполнения приложения для разного

числа перекрывающихся операций ввода и вывода (1, 2, 4, 8, 12, 16), не

забывая проверять работоспособность приложения (консольная команда FC).

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.

4. Подготовьте итоговый отчет с развернутыми выводами по Заданию 2. Управление дисками, каталогами и файлами

## 2.1. Вывод списка дисков

## Реализация вывода списка дисков с помощью функции GetLogicalDriveStrings();

## 

## Рисунок 1: Вывод списка дисков

## 2.2. Вывод информации о диске и размер свободного пространства

Для одного из выбранных дисков вывод информации о диске и размер свободного пространства (функции Win32 API – GetDriveType, GetVolumeInformation, GetDiskFreeSpace);

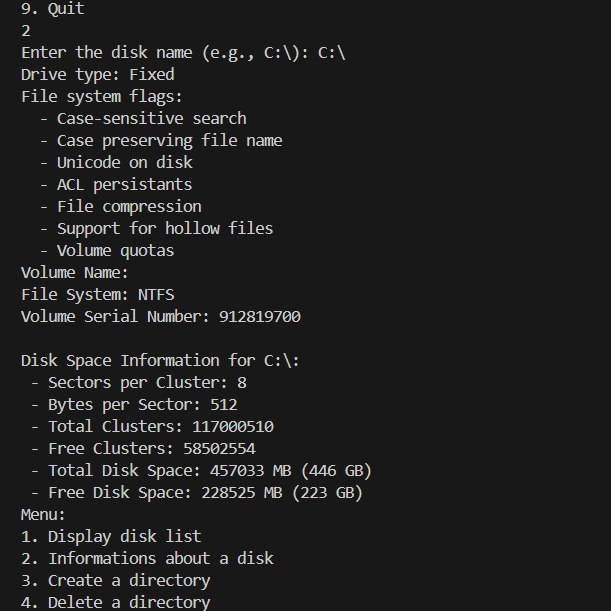


Рисунок 1: Вывод основной информации о дисках

## 2.3. Создание и удаление заданных каталогов

Реализация создания и удаления заданных каталогов с помощью функций CreateDirectory() и RemoveDirectory().

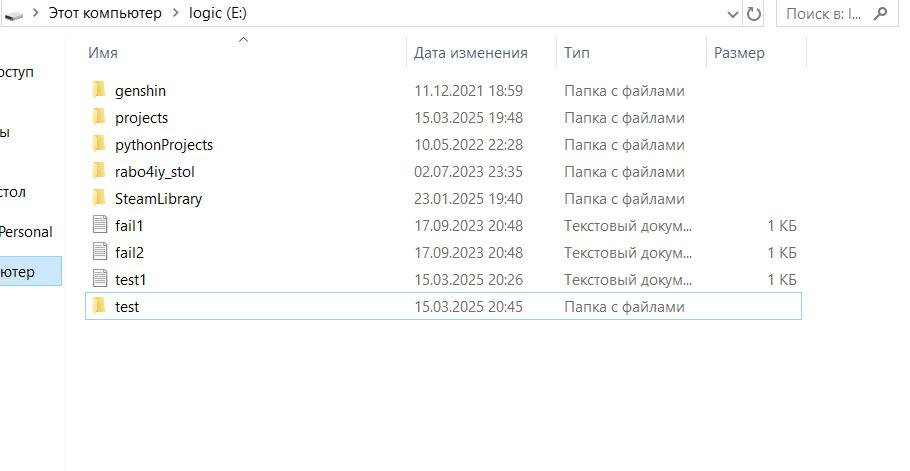
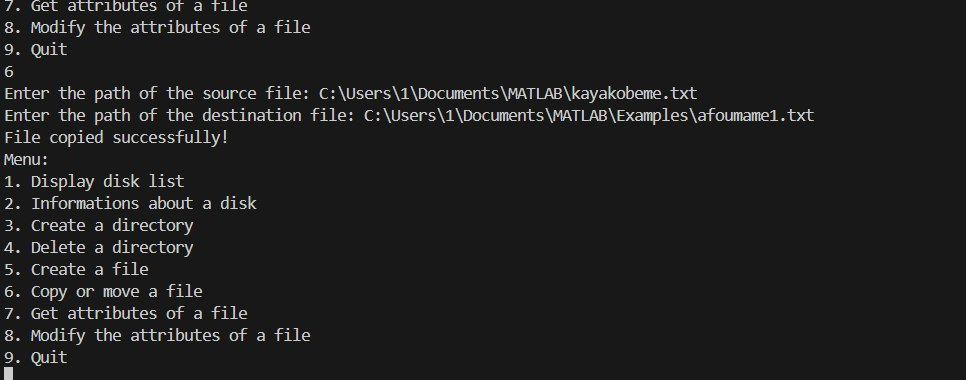


Рисунок 2: Создание каталога



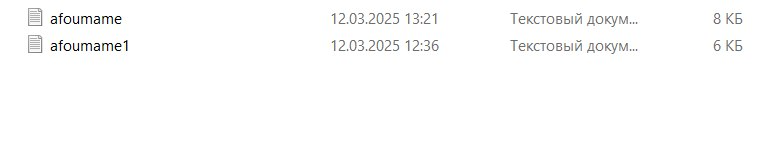


Рисунок 3: Директория каталога

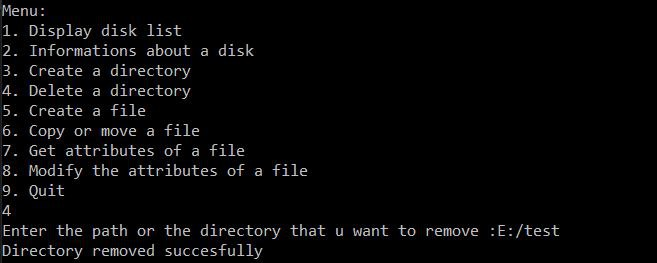


Рисунок 8: Удаление каталога

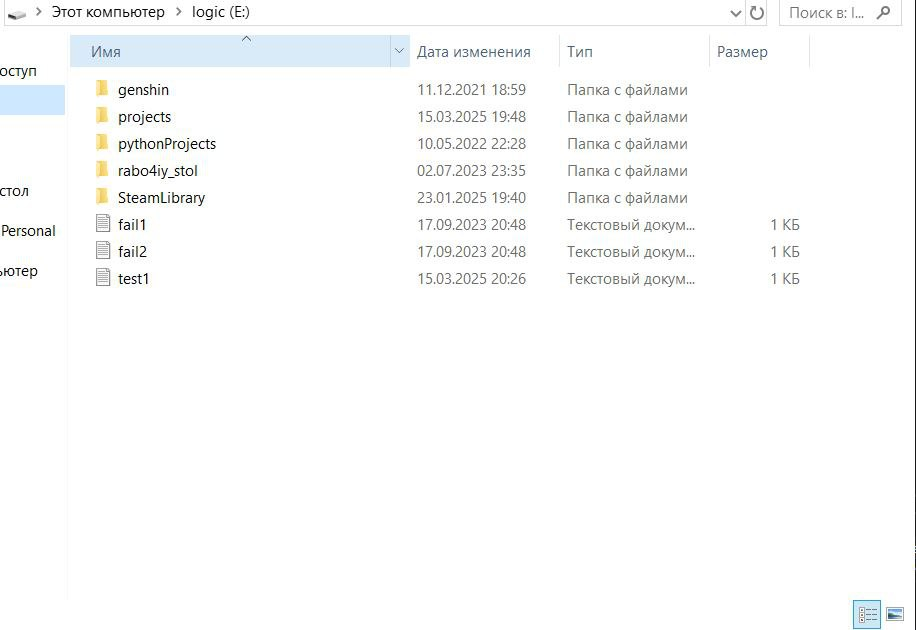


Рисунок 9: Директория каталога

## 2.4. Создание файлов в новых каталогах

Реализация создания файлов с помощью функции CreateFile().



Рисунок 10: Создание файла

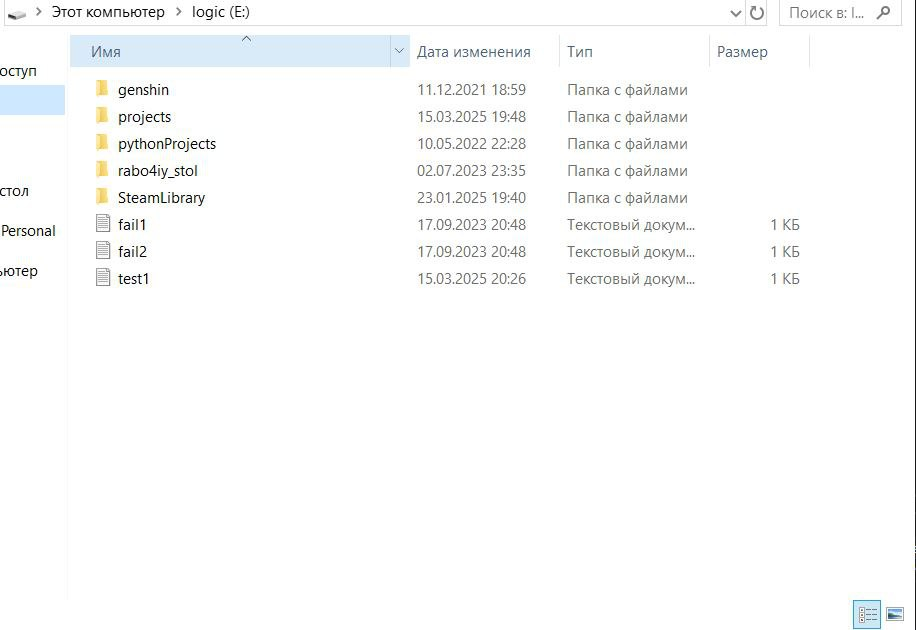


Рисунок 11: Директория файла

**2.5. Копирование и перемещение файлов между каталогами**

Реализация копирования и перемещения файлов между каталогами (в том числе с возможностью выбрать, как поведёт себя консольное приложение при возникновении файлов с одинаковым именем) с помощью функций CopyFile(), MoveFile() и MoveFileEx,

Рисунок 12: Копирование файла

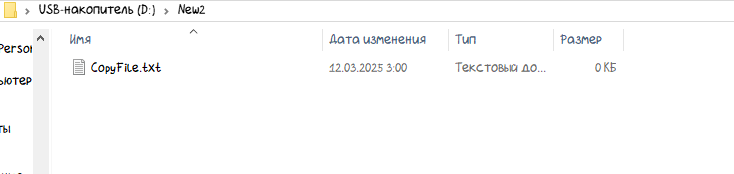


Рисунок 13: Директория после копирования

## 2.6. Анализ и изменение атрибутов файлов

Реализация анализа и изменения атрибутов файлов, а также их временных данных, получения данных с помощью дескриптора с помощью функций GetFileAttributes(), SetFileAttributes(), GetFileInformationByHandle(), GetFileTime() и SetFileTime().

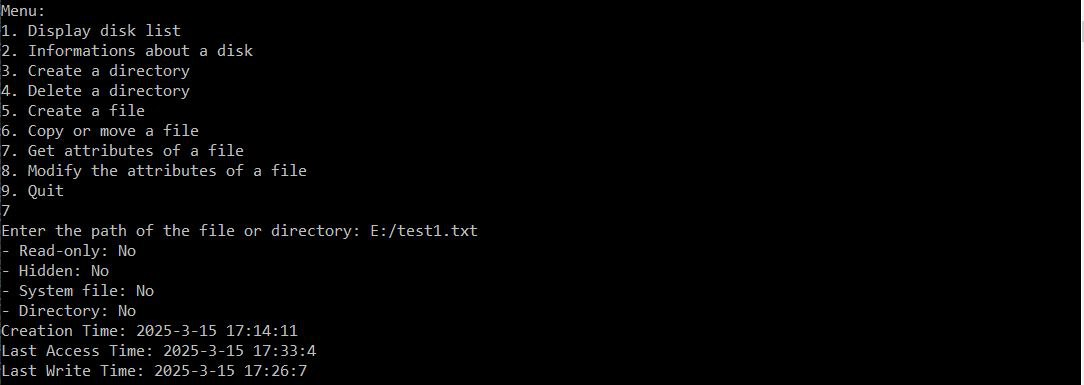


Рисунок 18: Вывод полученных данных о файле

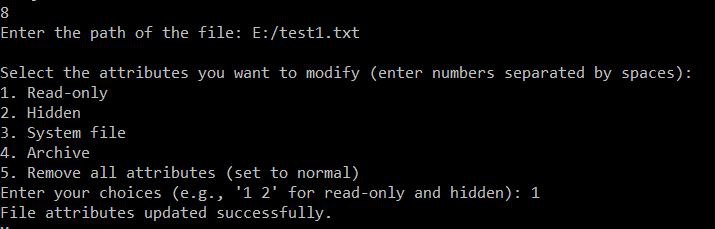


Рисунок 19: Изменение атрибутов

## 2.7. Исходный код программы

**main.cpp**:

|  |
| --- |
| #include <windows.h>  #include <iostream>  #include <string>  using namespace std;  const int MY\_MAX\_PATH = 260;  void listDrives() {  DWORD drives = GetLogicalDrives();  for (char drive = 'A'; drive <= 'Z'; drive++) {  if (drives & (1 << (drive -'A')))  {  std::cout << drive << ":\\";  }    }  cout << endl;  }  void diskInfo() {  string diskName;  cout << "Enter the disk name (e.g., C:\\): ";  getline(cin, diskName);  // Validation du format du nom du disque  if (diskName.empty() || diskName.back() != '\\') {  diskName += "\\";  }  int type = GetDriveType(diskName.c\_str());  switch (type) {  case DRIVE\_FIXED:  cout << "Drive type: Fixed" << endl;  break;  case DRIVE\_REMOVABLE:  cout << "Drive type: Removable" << endl;  break;  case DRIVE\_CDROM:  cout << "Drive type: CD-ROM" << endl;  break;  case DRIVE\_REMOTE:  cout << "Drive type: Remote" << endl;  break;  case DRIVE\_RAMDISK:  cout << "Drive type: RAM Disk" << endl;  break;  default:  cout << "Drive type: Unknown or invalid format..." << endl;  return;  }  char NameBuffer[MAX\_PATH];  char SysNameBuffer[MAX\_PATH];  DWORD VSNumber = 0;  DWORD MCLength = 0;  DWORD FileSF = 0;  if (!GetVolumeInformation(  diskName.c\_str(),  NameBuffer,  sizeof(NameBuffer),  &VSNumber,  &MCLength,  &FileSF,  SysNameBuffer,  sizeof(SysNameBuffer)  )) {  cerr << "Error: Unable to get volume information. Error code: " << GetLastError() << endl;  return;  }  std::wcout << L"File system flags: " << std::endl;  if (FileSF & FILE\_CASE\_SENSITIVE\_SEARCH) std::wcout << L" - Case-sensitive search" << std::endl;  if (FileSF & FILE\_CASE\_PRESERVED\_NAMES) std::wcout << L" - Case preserving file name" << std::endl;  if (FileSF & FILE\_UNICODE\_ON\_DISK) std::wcout << L" - Unicode on disk" << std::endl;  if (FileSF & FILE\_PERSISTENT\_ACLS) std::wcout << L" - ACL persistants" << std::endl;  if (FileSF & FILE\_FILE\_COMPRESSION) std::wcout << L" - File compression" << std::endl;  if (FileSF & FILE\_SUPPORTS\_SPARSE\_FILES) std::wcout << L" - Support for hollow files" << std::endl;  if (FileSF & FILE\_VOLUME\_QUOTAS) std::wcout << L" - Volume quotas" << std::endl;  cout << "Volume Name: " << NameBuffer << endl;  cout << "File System: " << SysNameBuffer << endl;  cout << "Volume Serial Number: " << VSNumber << endl;  DWORD sectorsPerCluster = 0; // Nombre de secteurs par cluster  DWORD bytesPerSector = 0; // Taille d'un secteur en octets  DWORD freeClusters = 0; // Nombre de clusters libres  DWORD totalClusters = 0; // Nombre total de clusters  BOOL success = GetDiskFreeSpaceA(  diskName.c\_str(), // Chemin du lecteur (e.g., "C:\\")  &sectorsPerCluster, // Nombre de secteurs par cluster  &bytesPerSector, // Taille d'un secteur en octets  &freeClusters, // Nombre de clusters libres  &totalClusters // Nombre total de clusters  );  if (!success) {  cerr << "Error: Unable to get disk space information. Error code: " << GetLastError() << endl;  return;  }  ULONGLONG totalBytes = static\_cast<ULONGLONG>(totalClusters) \* sectorsPerCluster \* bytesPerSector;  ULONGLONG freeBytes = static\_cast<ULONGLONG>(freeClusters) \* sectorsPerCluster \* bytesPerSector;  cout << "\nDisk Space Information for " << diskName << ":" << endl;  cout << " - Sectors per Cluster: " << sectorsPerCluster << endl;  cout << " - Bytes per Sector: " << bytesPerSector << endl;  cout << " - Total Clusters: " << totalClusters << endl;  cout << " - Free Clusters: " << freeClusters << endl;  cout << " - Total Disk Space: " << totalBytes / (1024 \* 1024) << " MB ("  << totalBytes / (1024ULL \* 1024 \* 1024) << " GB)" << endl;  cout << " - Free Disk Space: " << freeBytes / (1024 \* 1024) << " MB ("  << freeBytes / (1024ULL \* 1024 \* 1024) << " GB)" << endl;  }  /\*  void diskInfo() {  string diskName;  cout << "Enter the disk name :";  getline(cin, diskName);  int type = GetDriveType(diskName.c\_str());  wcout << L"Type de disque: ";  switch (type) {  case DRIVE\_UNKNOWN: wcout << L"UNKNOWN"; break;  case DRIVE\_NO\_ROOT\_DIR: wcout << L"No root repertory"; break;  case DRIVE\_REMOVABLE: wcout << L"Removable"; break;  case DRIVE\_FIXED: wcout << L"Fixed"; break;  case DRIVE\_REMOTE: wcout << L"Remote"; break;  case DRIVE\_CDROM: wcout << L"CD-ROM"; break;  case DRIVE\_RAMDISK: wcout << L"Ram Disk"; break;  default: wcout << L"Undefined"; break;  }  wcout << endl;  char NameBuffer[MAX\_PATH];  char SysNameBuffer[MAX\_PATH];  DWORD VSNumber;  DWORD MCLength;  DWORD FileSF;  if (GetVolumeInformationW(wpszProcToSearch, NameBuffer, sizeof(NameBuffer) / sizeof(wchar\_t),  &VSNumber, &maxComponentLength, &fileSystemFlags,  fileSystemNameBuffer, sizeof(fileSystemNameBuffer) / sizeof(wchar\_t))) {  std::wcout << L"Volum name: " << NameBuffer << std::endl;  std::wcout << L"Serie Number: " << VSNumber << std::endl;  std::wcout << L"File system: " << fileSystemNameBuffer << std::endl;  std::wcout << L"File system flags: " << std::endl;  if (fileSystemFlags & FILE\_CASE\_SENSITIVE\_SEARCH) std::wcout << L" - Case-sensitive search" << std::endl;  if (fileSystemFlags & FILE\_CASE\_PRESERVED\_NAMES) std::wcout << L" - Case preserving file name" << std::endl;  if (fileSystemFlags & FILE\_UNICODE\_ON\_DISK) std::wcout << L" - Unicode on disk" << std::endl;  if (fileSystemFlags & FILE\_PERSISTENT\_ACLS) std::wcout << L" - ACL persistants" << std::endl;  if (fileSystemFlags & FILE\_FILE\_COMPRESSION) std::wcout << L" - File compression" << std::endl;  if (fileSystemFlags & FILE\_SUPPORTS\_SPARSE\_FILES) std::wcout << L" - Support for hollow files" << std::endl;  if (fileSystemFlags & FILE\_VOLUME\_QUOTAS) std::wcout << L" - Volume quotas" << std::endl;  } else {  std::wcerr << L"Error while fetching volume information." << std::endl;  }  }  \*/  void createDirectory() {  string path;  cout << "Enter the path or the directory that u want to Create :";  getline(cin, path);  if (CreateDirectory(path.c\_str(), NULL)) {  cout << "Directory created successfully!" << endl;  } else{  cerr << "Error while creating the directory:" << GetLastError() << endl;  }    }  void deleteDirectory() {  string path;  cout << "Enter the path or the directory that u want to remove :";  getline(cin, path);  if (RemoveDirectoryA(path.c\_str()))  {  cout << "Directory removed succesfully";  }else {  cout << "Error while removing the directory: " << GetLastError() << endl;  }  }  void createFile() {  string filePath;  cout << "Enter the path and the name of the file you want to create (C:\\MyFolder\\myfile.txt)";  getline(cin, filePath);  HANDLE hFile = CreateFileA(  filePath.c\_str(), //Chemin du fichier  GENERIC\_WRITE, //Autorisation d'ecriture  0,  NULL, //Attributs de securite par defaut  CREATE\_ALWAYS, //Cree toujours  FILE\_ATTRIBUTE\_NORMAL, //Attribut normaux  NULL //Aucun modele de fichier  );  if (hFile == INVALID\_HANDLE\_VALUE) {  cout << "Error while creating file" << GetLastError() << endl;  } else {  cout << "File created succesfully :" << filePath << endl;  CloseHandle(hFile);  }  }  void copyFile() {  string sourcePath, destinationPath;  cout << "Enter the path of the source file : ";  cin.ignore();  getline(cin, sourcePath);  cout << "Enter the path of the destination file : ";  cin.ignore();  getline(cin, destinationPath);  if (sourcePath.empty() || destinationPath.empty())  {  cout << "Error: Source and destination paths cannot be empty." << endl;  }    bool success = CopyFileA(  sourcePath.c\_str(),  destinationPath.c\_str(),  false //true: Echoue si le fichier de destination existe deja  //false: Ecrase le fichier de destination s'il existe deja  );  if (success) {  cout << "File copied successfully !" << endl;  } else {  cout << "Operation failed!";  }  }  void getFileAttributes() {  string filePath;  cout << "Enter the path of the source file : ";  cin.ignore();  getline(cin, filePath);  DWORD attributes = GetFileAttributesA(filePath.c\_str());  if (attributes == INVALID\_FILE\_ATTRIBUTES) {  cout << " - Read-only: Yes " << endl;  }else{  cout << " - Read-only: No " << endl;  }  if (attributes & FILE\_ATTRIBUTE\_HIDDEN) {  std::cout << " - Hidden: Yes" << std::endl;  } else {  std::cout << " - Hidden: No" << std::endl;  }  if (attributes & FILE\_ATTRIBUTE\_SYSTEM) {  std::cout << " - System file: Yes" << std::endl;  } else {  std::cout << " - System file: No" << std::endl;  }  if (attributes & FILE\_ATTRIBUTE\_DIRECTORY) {  std::cout << " - Directory: Yes" << std::endl;  } else {  std::cout << " - Directory: No" << std::endl;  }  HANDLE hFile = CreateFileA(  filePath.c\_str(),  GENERIC\_READ,  FILE\_SHARE\_READ,  NULL,  OPEN\_EXISTING,  FILE\_ATTRIBUTE\_NORMAL,  NULL  );  if (hFile == INVALID\_HANDLE\_VALUE)  {  cerr <<"Error: Unable to open file. Error code: " << GetLastError() << endl;  }  BY\_HANDLE\_FILE\_INFORMATION fileInfo;  if (!GetFileInformationByHandle(hFile, &fileInfo))  {  cerr << "Error: Unable to get file time. Error code: " << GetLastError() << endl;  CloseHandle(hFile);  return;  }  SYSTEMTIME sysCreationTime, sysLastAccessTime, sysLastWriteTime;  FileTimeToSystemTime(&fileInfo.ftCreationTime, &sysCreationTime);  FileTimeToSystemTime(&fileInfo.ftLastAccessTime, &sysLastAccessTime);  FileTimeToSystemTime(&fileInfo.ftLastWriteTime, &sysLastWriteTime);  std::cout << "Creation Time: " << sysCreationTime.wYear << "-"  << sysCreationTime.wMonth << "-" << sysCreationTime.wDay << " "  << sysCreationTime.wHour << ":" << sysCreationTime.wMinute << ":"  << sysCreationTime.wSecond << std::endl;  std::cout << "Last Access Time: " << sysLastAccessTime.wYear << "-"  << sysLastAccessTime.wMonth << "-" << sysLastAccessTime.wDay << " "  << sysLastAccessTime.wHour << ":" << sysLastAccessTime.wMinute << ":"  << sysLastAccessTime.wSecond << std::endl;  std::cout << "Last Write Time: " << sysLastWriteTime.wYear << "-"  << sysLastWriteTime.wMonth << "-" << sysLastWriteTime.wDay << " "  << sysLastWriteTime.wHour << ":" << sysLastWriteTime.wMinute << ":"  << sysLastWriteTime.wSecond << std::endl;  CloseHandle(hFile);    }  void updateAttributes() {  string filePath;  cout << "Enter the path of the file : ";  cin.ignore();  getline(cin, filePath);  DWORD attributes = GetFileAttributesA(filePath.c\_str());  if (attributes == INVALID\_FILE\_ATTRIBUTES) {  std::cerr << "Error: Unable to get file attributes. Error code: " << GetLastError() << std::endl;  return;  }  std::cout << "\nSelect the attributes you want to modify (enter numbers separated by spaces):" << std::endl;  std::cout << "1. Read-only" << std::endl;  std::cout << "2. Hidden" << std::endl;  std::cout << "3. System file" << std::endl;  std::cout << "4. Archive" << std::endl;  std::cout << "5. Remove all attributes (set to normal)" << std::endl;  std::cout << "Enter your choices (e.g., '1 2' for read-only and hidden): ";  std::string input;  std::getline(std::cin, input);  bool modified = false;  size\_t pos = 0;  while (pos < input.length()) {  size\_t nextPos = input.find(' ', pos);  std::string choiceStr = input.substr(pos, nextPos - pos);  int choice = std::stoi(choiceStr);  switch (choice) {  case 1: // Read-only  attributes |= FILE\_ATTRIBUTE\_READONLY;  modified = true;  break;  case 2: // Hidden  attributes |= FILE\_ATTRIBUTE\_HIDDEN;  modified = true;  break;  case 3: // System file  attributes |= FILE\_ATTRIBUTE\_SYSTEM;  modified = true;  break;  case 4: // Archive  attributes |= FILE\_ATTRIBUTE\_ARCHIVE;  modified = true;  break;  case 5: // Remove all attributes (set to normal)  attributes = FILE\_ATTRIBUTE\_NORMAL;  modified = true;  break;  default:  std::cerr << "Invalid choice: " << choice << ". Skipping." << std::endl;  break;  }  pos = (nextPos == std::string::npos) ? input.length() : nextPos + 1;  }  if (modified) {  BOOL success = SetFileAttributesA(filePath.c\_str(), attributes);  if (!success) {  std::cerr << "Error: Unable to set file attributes. Error code: " << GetLastError() << std::endl;  } else {  std::cout << "File attributes updated successfully." << std::endl;  }  } else {  std::cout << "No changes were made to the file attributes." << std::endl;  }  }  int main() {  int choice;  do  {  cout << "Menu:" << endl;  cout << "1. Display disk list" << endl;  cout << "2. Informations about a disk" << endl;  cout << "3. Create a directory" << endl;  cout << "4. Delete a directory" << endl;  cout << "5. Create a file" << endl;  cout << "6. Copy or move a file" << endl;  cout << "7. Get attributes of a file" << endl;  cout << "8. Modify the attributes of a file" << endl;  cout << "9. Quit" << endl;  cin >> choice;  cin.ignore(); // Pour ignorer le retour à la ligne après la saisie du choix  switch (choice)  {  case 1:  listDrives();  break;  case 2: {  diskInfo();  break;  }  case 3: {  createDirectory();  break;  }  case 4: {  deleteDirectory();  break;  }  case 5: {  createFile();  break;  }  case 6: {  copyFile();  break;  }  case 7: {  getFileAttributes();  break;  }  case 8: {  updateAttributes();  break;  }  case 9: {  break;  }  default: {  break;  }  }  } while (choice != 9);    return 0;  } |

## 2.8. Выводы

В рамках этого проекта было разработано консольное приложение с меню, позволяющее управлять дисками, каталогами и файлами с помощью функций Win32 API. Приложение предлагает такие функции, как просмотр списка дисков, получение информации о них, создание и удаление каталогов, а также управление файлами (создание, копирование, перемещение). Она также позволяет анализировать и изменять атрибуты файлов. Проверочные тесты подтвердили, что функции Win32 API работают нормально, обеспечивая эффективное управление файловой системой в Windows.

# 3. Копирование файла с помощью перекрывающих операций ввода-вывода

## 3.1. Создание и запуск консольного приложения

## 

Рисунок 24: Компиляция программы, тестовый запуск на файле в 16.8МБ, проверка работоспособности

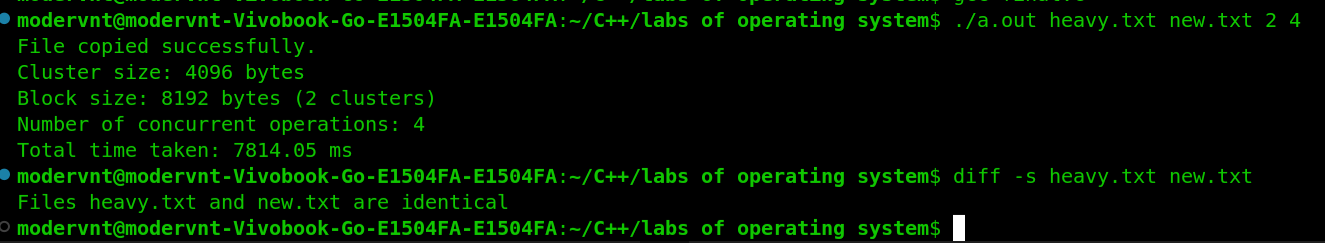


Рисунок 4: Использование команды diff для проверки результата копирования

## 3.2. Проверка приложения на разных размерах копируемых блоков

Результаты показывают значительное улучшение производительности по мере увеличения размеров блоков, используемых для асинхронных операций ввода-вывода. При использовании блоков небольшого размера (1024 или 2048 байт) время выполнения остается высоким из-за большого количества необходимых операций и накладных расходов, связанных с системными вызовами. Однако по мере увеличения размеров блоков (до 32768 байт или более) время выполнения резко сокращается, достигая минимума в 2372,79 мс при размере блока 128 Мб. Эти наблюдения подчеркивают важность выбора подходящего размера блока : слишком маленький размер приводит к значительным накладным расходам ; слишком большой размер может снизить эффективность из-за аппаратных ограничений. Размер блока около 32 КБ, по-видимому, является хорошим компромиссом, оптимизирующим как скорость, так и использование ресурсов.

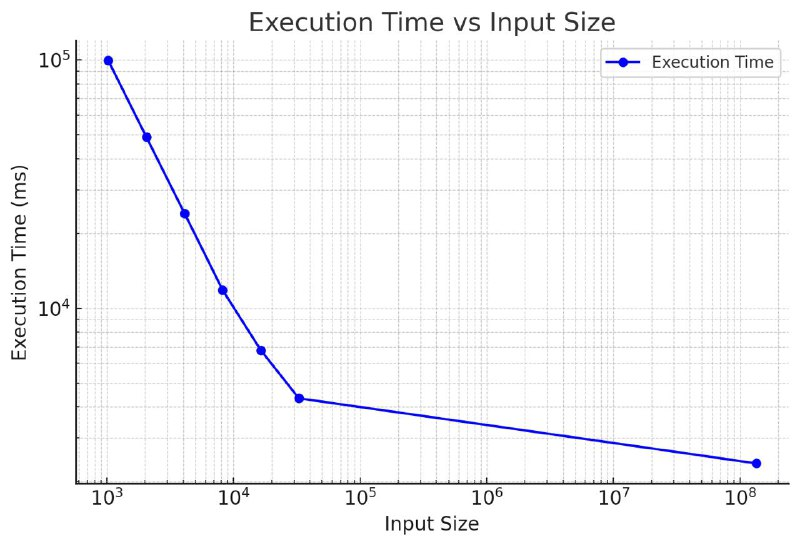


Рисунок 5: Зависимость времени от множителя размера кластера

## 3.3. Проверка приложения на разном числе операция ввода-вывода

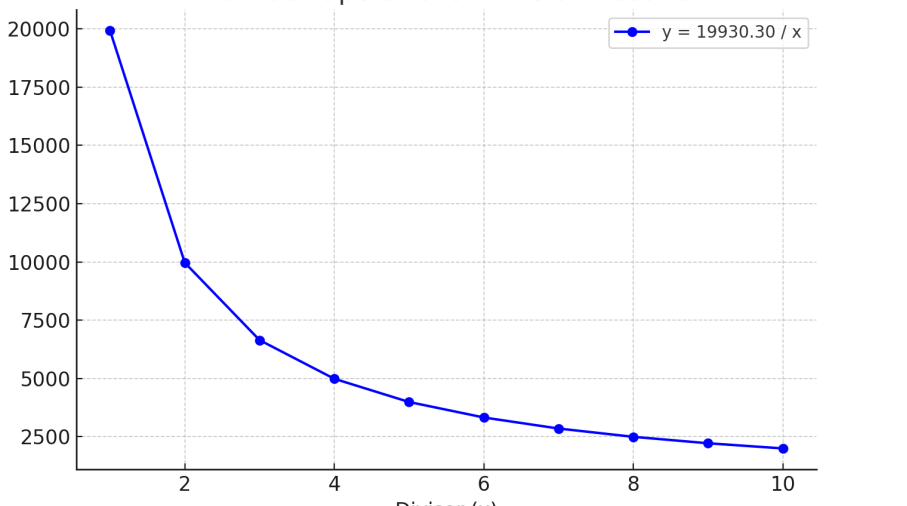


Рисунок 6: Зависимость времени от

числа перекрывающихся операций ввода и вывода

это исследование помогло продемонстрировать важность параллелизма в асинхронных операциях копирования, а также пролило свет на ограничения, налагаемые аппаратным обеспечением и операционной системой. Оптимальное количество одновременных операций для копирования файла размером 1 Гб в тестируемую конфигурацию - 6, при этом время выполнения достигает минимума (14230,57 МС). эти результаты подчеркивают необходимость достижения баланса между параллелизмом и управлением ресурсами для максимизации производительности. Будущие исследования и оптимизация могут еще больше улучшить эти результаты с учетом особенностей оборудования и доступных новых технологий.

## 3.4. Исходный код программы

**main.cpp**

|  |
| --- |
| #define \_GNU\_SOURCE  #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h> // open  #include <unistd.h> // close, fstat, access  #include <aio.h> // aio\_read, aio\_write  #include <sys/stat.h> // fstat  #include <sys/statvfs.h> // statvfs  #include <string.h> // strerror  #include <time.h> // clock\_gettime  #include <errno.h> // errno  #include <signal.h> // sigval\_t  // Structure pour gérer les opérations asynchrones  typedef struct async\_op {  struct aiocb aio; // Structure pour les opérations asynchrones  void\* data\_buffer; // Buffer pour les données (aligné)  int is\_write; // 1 pour écriture, 0 pour lecture  } async\_op;  // Gestionnaire de fin d'opération asynchrone  void handle\_completion(sigval\_t signal) {  async\_op\* operation = (async\_op\*)signal.sival\_ptr;  if (operation->is\_write) {  ssize\_t bytes\_written = aio\_return(&operation->aio);  if (bytes\_written == -1) {  fprintf(stderr, "Write operation failed: %s T\_T\n", strerror(errno));  }  } else {  ssize\_t bytes\_read = aio\_return(&operation->aio);  if (bytes\_read == -1) {  fprintf(stderr, "Read operation failed: %s T\_T\n", strerror(errno));  }  }  }  // Créer un fichier et le remplir de données aléatoires  void generate\_random\_file(const char\* filename, size\_t size) {  int file\_descriptor = open(filename, O\_CREAT | O\_WRONLY | O\_TRUNC, 0666);  if (file\_descriptor == -1) {  fprintf(stderr, "Failed to create the file: %s T\_T\n", strerror(errno));  exit(EXIT\_FAILURE);  }  char\* buffer = malloc(size);  if (!buffer) {  fprintf(stderr, "Memory allocation failed T\_T\n");  close(file\_descriptor);  exit(EXIT\_FAILURE);  }  for (size\_t i = 0; i < size; ++i) {  buffer[i] = 'A' + rand() % ('Z' - 'A' + 1); // Lettres majuscules aléatoires  }  size\_t total\_written = 0;  while (total\_written < size) {  ssize\_t bytes\_written = write(file\_descriptor, buffer + total\_written, size - total\_written);  if (bytes\_written == -1) {  fprintf(stderr, "Failed to write to file: %s T\_T\n", strerror(errno));  free(buffer);  close(file\_descriptor);  exit(EXIT\_FAILURE);  }  total\_written += bytes\_written;  }  free(buffer);  close(file\_descriptor);  printf("File created and filled with random data: %s\n", filename);  }  // Copie asynchrone principale  void perform\_async\_copy(const char\* input\_file, const char\* output\_file, size\_t block\_size, int num\_ops) {  // Vérifier si le fichier source existe  if (access(input\_file, F\_OK) == -1) {  printf("Source file does not exist. Creating and filling with random data...\n");  size\_t default\_size = 64 \* 1024 \* 1024; // 64 Mo par défaut  generate\_random\_file(input\_file, default\_size);  }  // Ouvrir le fichier source en lecture  int read\_fd = open(input\_file, O\_RDONLY);  if (read\_fd == -1) {  fprintf(stderr, "Failed to open source file: %s T\_T\n", strerror(errno));  exit(EXIT\_FAILURE);  }  // Ouvrir le fichier de destination en écriture  int write\_fd = open(output\_file, O\_CREAT | O\_WRONLY | O\_TRUNC, 0666);  if (write\_fd == -1) {  close(read\_fd);  fprintf(stderr, "Failed to open destination file: %s T\_T\n", strerror(errno));  exit(EXIT\_FAILURE);  }  // Obtenir la taille du fichier source  struct stat file\_info;  if (fstat(read\_fd, &file\_info) == -1) {  close(read\_fd);  close(write\_fd);  fprintf(stderr, "Failed to get file size: %s T\_T\n", strerror(errno));  exit(EXIT\_FAILURE);  }  off\_t file\_size = file\_info.st\_size;  off\_t remaining\_bytes = file\_size;  // Initialiser le pool d'opérations asynchrones  async\_op operations[num\_ops];  memset(operations, 0, sizeof(async\_op) \* num\_ops);  for (int i = 0; i < num\_ops; ++i) {  if (posix\_memalign(&operations[i].data\_buffer, block\_size, block\_size) != 0) {  fprintf(stderr, "Failed to allocate aligned buffer T\_T\n");  exit(EXIT\_FAILURE);  }  }  // Positions pour lecture/écriture  off\_t read\_pos = 0;  off\_t write\_pos = 0;  // Mesurer le temps d'exécution  struct timespec start\_time, end\_time;  clock\_gettime(CLOCK\_MONOTONIC, &start\_time);  // Boucle de copie asynchrone  while (remaining\_bytes > 0) {  // Lancer les opérations de lecture  for (int i = 0; i < num\_ops && remaining\_bytes > 0; ++i) {  async\_op\* op = &operations[i];  op->is\_write = 0;  op->aio.aio\_fildes = read\_fd;  op->aio.aio\_buf = op->data\_buffer;  op->aio.aio\_nbytes = remaining\_bytes < block\_size ? remaining\_bytes : block\_size;  op->aio.aio\_offset = read\_pos;  op->aio.aio\_sigevent.sigev\_notify = SIGEV\_THREAD;  op->aio.aio\_sigevent.sigev\_notify\_function = handle\_completion;  op->aio.aio\_sigevent.sigev\_value.sival\_ptr = op;  if (aio\_read(&op->aio) == -1) {  fprintf(stderr, "Failed to initiate read operation: %s T\_T\n", strerror(errno));  exit(EXIT\_FAILURE);  }  read\_pos += op->aio.aio\_nbytes;  remaining\_bytes -= op->aio.aio\_nbytes;  }  // Attendre la fin des lectures  for (int i = 0; i < num\_ops; ++i) {  while (aio\_error(&operations[i].aio) == EINPROGRESS) {  // Attendre  }  ssize\_t bytes\_read = aio\_return(&operations[i].aio);  if (bytes\_read == -1) {  fprintf(stderr, "Read operation failed: %s T\_T\n", strerror(errno));  exit(EXIT\_FAILURE);  }  // Lancer l'écriture  operations[i].is\_write = 1;  operations[i].aio.aio\_fildes = write\_fd;  operations[i].aio.aio\_offset = write\_pos;  if (aio\_write(&operations[i].aio) == -1) {  fprintf(stderr, "Failed to initiate write operation: %s T\_T\n", strerror(errno));  exit(EXIT\_FAILURE);  }  write\_pos += bytes\_read;  }  // Attendre la fin des écritures  for (int i = 0; i < num\_ops; ++i) {  while (aio\_error(&operations[i].aio) == EINPROGRESS) {  // Attendre  }  ssize\_t bytes\_written = aio\_return(&operations[i].aio);  if (bytes\_written == -1) {  fprintf(stderr, "Write operation failed: %s T\_T\n", strerror(errno));  exit(EXIT\_FAILURE);  }  }  }  // Libérer les buffers  for (int i = 0; i < num\_ops; ++i) {  free(operations[i].data\_buffer);  }  close(read\_fd);  close(write\_fd);  // Calculer le temps total  clock\_gettime(CLOCK\_MONOTONIC, &end\_time);  double duration = (end\_time.tv\_sec - start\_time.tv\_sec) \* 1000.0 +  (end\_time.tv\_nsec - start\_time.tv\_nsec) / 1000000.0;  printf("File copied successfully.\n");  //printf("Cluster size: %zu bytes\n", cluster\_size);  printf("Block size: %zu bytes \n", block\_size);  printf("Number of concurrent operations: %d\n", num\_ops);  printf("Total time taken: %.2f ms\n", duration);  }  int main(int argc, char\*\* argv) {  if (argc != 5) {  fprintf(stderr, "Usage: %s <source\_file> <destination\_file> <cluster\_multiplier> <num\_operations>\n", argv[0]);  return EXIT\_FAILURE;  }  const char\* source\_file = argv[1];  const char\* dest\_file = argv[2];  size\_t block\_size = strtoul(argv[3], NULL, 10);  int num\_operations = atoi(argv[4]);  int write\_fd = open(dest\_file, O\_CREAT | O\_WRONLY | O\_TRUNC, 0666);  if (write\_fd == -1)  {  fprintf(stderr, "Failed to open or create destination file '%s': %s T\_T\n", dest\_file, strerror(errno));  exit(EXIT\_FAILURE);  }  perform\_async\_copy(source\_file, dest\_file, block\_size, num\_operations);  return EXIT\_SUCCESS;  } |

## 3.5. Выводы

В заключение, выбор размера блока является ключевым параметром для оптимизации производительности операций асинхронного ввода-вывода. Тщательный анализ и тестирование, адаптированные к конкретной среде, необходимы для определения оптимальной конфигурации, тем самым гарантируя эффективное использование ресурсов при минимальном времени обработки. Эти наблюдения дают ценные рекомендации по улучшению приложений, требующих интенсивной передачи данных, с учетом аппаратных и программных ограничений.