Белорусский государственный технологический университет
Факультет информационных технологий
Кафедра программной инженерии

Лабораторная работа 9

По дисциплине «Операционные системы»

На тему «Управление вводом/выводом»

Выполнил:
Студент 3 курса 9 группы
Павлович Ян Андреевич
Преподаватель: Савельева М.Г.

Минск, 2025

Введение

Целью работы является освоение практических навыков работы с подсистемой ввода/вывода в операционных системах Windows и Linux.

В рамках работы необходимо разработать консольные приложения для Windows и Linux, демонстрирующие работу с механизмами ввода-вывода, управление файлами и каталогами, а также синхронизацию потоков и процессов. Программы должны выполнять операции открытия, чтения, записи и копирования файлов, отображать информацию о каталогах и отслеживать изменения в файловой системе.

Для платформы Windows требуется использовать WinAPI для работы с файлами и каталогами, реализовать асинхронное копирование файлов с помощью OVERLAPPED I/O и потоков, управлять ресурсами, обрабатывать сигналы через системные обработчики и выводить статистику копирования и состояние операций. На платформе Linux создаются аналогичные приложения с применением POSIX API, системных вызовов open, read, write, stat, opendir, readdir, многопоточного копирования файлов через POSIX Threads и вызовов pread/pwrite, мониторинга изменений в каталогах с помощью inotify и обработки сигналов для остановки асинхронных операций.

В ходе работы необходимо провести сравнительный анализ поведения приложений на Windows и Linux, выявить различия в реализации асинхронного ввода-вывода, мониторинга каталогов и многопоточной обработки, а также оценить эффективность используемых механизмов и влияние синхронизации и сигналов на производительность и устойчивость программ.

Используемые инструменты:

- Parallels Desktop
- Командная оболочка cmd
- Process Explorer
- компилятор *g++*
- среда разработки Visual Studio

1 Windows

1.1 Приложение Lab-09a

В рамках данной лабораторной работы необходимо создать консольное приложение Lab-09a, в котором реализованы две основные функции: PrintInfo(LPSTR FileName) и PrintText(LPSTR FileName).

Функция PrintInfo предназначена для детального анализа файла. Она выводит на экран имя файла, его размер в различных единицах измерения, а также определяет тип файла (что отличается от его расширения). Помимо этого, функция отображает время создания файла, время последнего доступа и время последнего изменения. В случае бинарных файлов PrintInfo дополнительно указывает, к какому типу они относятся. Функция PrintText отвечает за отображение содержимого файла в консоли. Если файл не является текстовым, функция информирует об этом пользователя и завершает свое выполнение.

Путь к файлу передается через аргументы командной строки, что позволяет использовать приложение с любыми файлами без необходимости изменения кода. Особое внимание уделено обработке ошибок: приложение проверяет корректность переданных аргументов, наличие файла и права доступа к нему, а также корректно освобождает все ресурсы после завершения работы функций.

```
C:\Lab-09\x64\Debug>Lab-09a.exe A.txt
File name: A.txt
File size: 3306 bytes (3.23 KiB, 0.00 MiB)
Type: regular file
Creation time: 30.12.2025 18:04
Last access time: 30.12.2025 18:10
Last modification time: 30.12.2025 18:10
The file is a text file

--- File content ---
#include <windows.h>
#include <stdio.h>
#include <locale.h>

BOOL IsTextFile(LPCSTR FileName) {
    HANDLE hFile = CreateFileA(FileName, GENERIC_READ, FILE_SHARE_READ, NULL,
        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == INVALID_HANDLE_VALUE) return FALSE;

    BYTE buffer[512];
    DWORD bytesRead;
    BOOL isText = TRUE;

    if (ReadFile(hFile, buffer, sizeof(buffer), &bytesRead, NULL)) {
        for (DWORD i = 0; i < bytesRead; i++) {
            if (buffer[i] < 9 || (buffer[i] > 13 && buffer[i] < 32)) {
                isText = FALSE;
```

Рисунок 1.1 – Результат работы Lab-09a

Для работы приложения создается консольное приложение, в которое заносится код, представленный в листинге А. После компиляции и запуска программа выводит информацию о выбранном файле и, при необходимости, его содержимое. Пример работы приложения показан на рисунке 1.1, где наглядно демонстрируется вывод функций PrintInfo и PrintText.

1.2 Приложение Lab-09b

Для выполнения данной лабораторной работы было разработано консольное приложение Lab-09b, предназначенное для работы с текстовым файлом, содержащим список студентов. В приложении реализованы функции OpenFile(LPSTR filePath), AddRow(HANDLE hFile, LPSTR row, INT pos), RemRow(HANDLE hFile, INT pos), PrintRow(HANDLE hFile, INT pos), PrintRows(HANDLE hFile) и CloseFile(HANDLE hFile).

Хранение дескриптора открытого файла и всех необходимых буферов для работы с данными организовано на глобальном уровне с помощью глобальных переменных. Взаимодействие с файлом осуществляется исключительно через вышеуказанные функции, а вызовы функций пользователем реализованы через консольное меню, что позволяет вводить необходимые команды и данные прямо с клавиатуры. В приложении предусмотрена обработка ошибок ввода данных, а также проверка корректности всех операций.

Функция OpenFile открывает указанный файл и сохраняет его дескриптор в глобальной переменной. Если файл не существует, возвращается ошибка. Функция AddRow добавляет строку в файл на указанную позицию: при pos > 0 строка вставляется в соответствующее место (отсчет начинается с 1), при pos = 0 — в начало файла, при pos = -1 — в конец файла. В остальных случаях или при некорректном дескрипторе добавление не выполняется, возвращая ошибку.

Функция RemRow удаляет строку из файла по аналогичным правилам: при pos > 0 удаляется указанная строка, при pos = 0 — первая, при pos = -1 — последняя, в остальных случаях или при некорректном дескрипторе возвращается ошибка.

Функция PrintRow выводит на консоль строку с заданной позиции с теми же условиями.

Функция PrintRows отображает все содержимое файла, при отсутствии корректного дескриптора возвращается ошибка.

Наконец, CloseFile переводит дескриптор файла в недействительное состояние и освобождает все динамически выделенные ресурсы; при отсутствии корректного дескриптора также возвращается ошибка.

```
Select an operation:
1. Open file
2. Insert row
3. Remove row
4. Print row
5. Print file
6. Close file
0. Exit
> 1
Enter file path: C:\Lab-09\x64\Debug\A.txt
File opened. Lines: 49, Size: 809 bytes.
```

Рисунок 1.2 – Результат второй программы опции 1

```
> 2
Enter insert position (0 = start, -1 = end, >0 = index): 0
Enter row text: Yan Pavlovich
Row added at position 0.

Select an operation:
1. Open file
2. Insert row
3. Remove row
4. Print row
5. Print file
6. Close file
0. Exit
> 5
1. Yan Pavlovich
2. Alexander Ivanov
3. Dmitry Petrov
```

Рисунок 1.3 – Результат второй программы опции 2 и 5

```
> 3
Enter delete position (0 = first, -1 = last, >0 = index): 0
Row removed from position 0.

Select an operation:
1. Open file
2. Insert row
3. Remove row
4. Print row
5. Print file
6. Close file
0. Exit
> 5
1. Alexander Ivanov
2. Dmitry Petrov
3. Anna Sidorova
4. Maxim Kuznetsov
```

Рисунок 1.4 – Результат второй программы опции 3

```
> 4
Enter print position (0 = first, -1 = last, >0 = index): 0
Row [1]: Alexander Ivanov
```

Рисунок 1.5 – Результат второй программы опции 4

```
0. Exit
> 6
File closed, resources released.
```

Рисунок 1.6 – Результат второй программы опции 6

```
6. Close file
0. Exit
> 0

C:\Lab-09\x64\Debug>
```

Рисунок 1.7 – Результат второй программы опции 0

Во время работы приложения можно наблюдать состояние файла в Process Explorer: после открытия и после его закрытия. После открытия файла он появляется в свойствах приложения, а после закрытия корректно удаляется из них. Результаты представлены на рисунках 1.8-1.9.

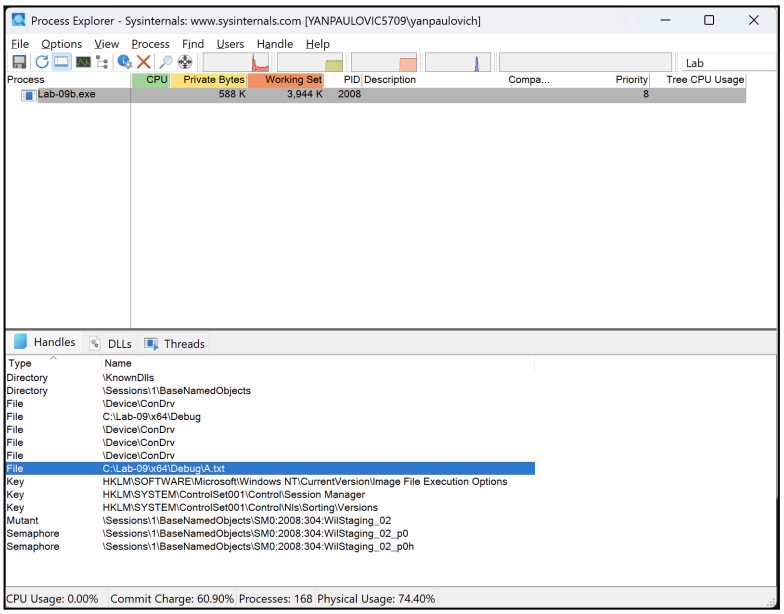

Рисунок 1.8 – Результат Process Explorer с открытым файлом

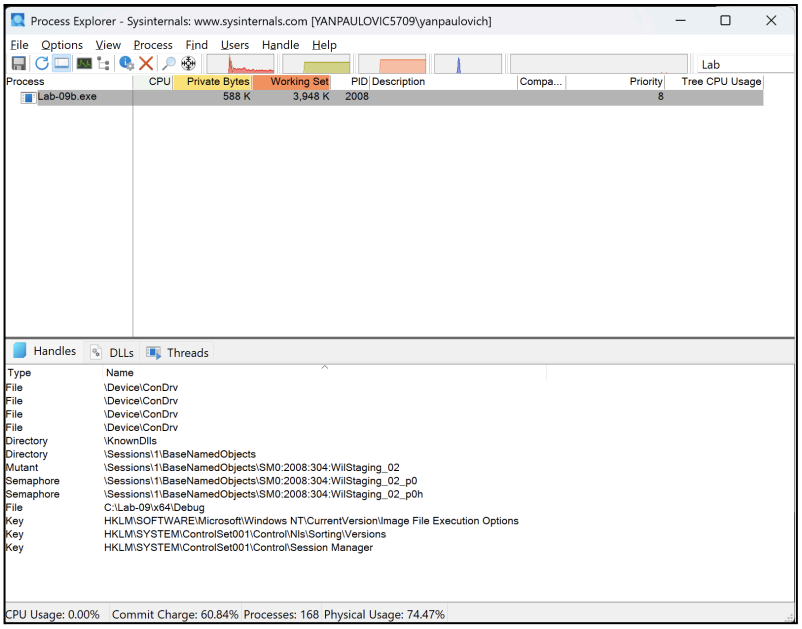

Рисунок 1.9 – Результат Process Explorer с закрытым файлом

Консольное приложение создается и в него записывается код, представленный в листинге Б. После компиляции и запуска программы выполняются все операции с файлом, включая открытие, добавление строки, считывание всего файла, вывод отдельных строк, удаление строк и закрытие файла с освобождением ресурсов. Результаты работы программы наглядно показаны на рисунках 1.2–1.7.

1.3 Приложение Lab-09c

В рамках лабораторной работы было разработано консольное приложение Lab-09c, основанное на использовании функции ReadDirectoryChangesW. Приложение предназначено для анализа содержимого каталога и мониторинга изменений в файловой системе.

На начальном этапе программа выводит список элементов указанного каталога, включая файлы и вложенные каталоги только первого уровня. Если заданный каталог не существует или недоступен, приложение сообщает об ошибке и завершает свою работу. Путь к отслеживаемому каталогу передается через аргументы командной строки. После вывода содержимого каталога запускается бесконечный цикл, в рамках которого выполняется отслеживание изменений в выбранной директории. При возникновении любого события, связанного с файловой системой, приложение выводит в консоль тип произошедшего события.

```
C:\Lab-09\x64\Debug>Lab-09c.exe C:\Lab-09\x64\Debug
Contents of directory C:\Lab-09\x64\Debug:
[FILE] A.txt
[FILE] Lab-09a.exe
[FILE] Lab-09a.pdb
[FILE] Lab-09b.exe
[FILE] Lab-09b.pdb
[FILE] Lab-09c.exe
[FILE] Lab-09c.pdb

Monitoring changes in directory C:\Lab-09\x64\Debug...
Added: New Text Document.txt
Renamed (old name): New Text Document.txt
Removed: B.txt
Modified: A.txt
^C
C:\Lab-09\x64\Debug>
```

Рисунок 1.10 – Результат работы Lab-09c

Для реализации функциональности создается консольное приложение, в которое добавляется код, представленный в листинге В. После компиляции и запуска программы выполняется ряд действий в отслеживаемом каталоге, результаты которых отображаются в консоли, что показано на рисунке 1.10.

В процессе работы приложения каждое событие сопровождается уведомлением в консоли.

2 Linux

2.1 Приложение Lab-09a

В рамках работы было разработано консольное приложение, являющееся функциональным аналогом приложения Lab-09a, реализованного для платформы Windows. Основное отличие заключается в отсутствии механизма определения того, является ли файл бинарным, так как данный пункт исключен из требований для версии под Linux.

Для выполнения задания создается и компилируется консольное приложение, в которое включается код, представленный в листинге Г. После сборки программа запускается и выводит информацию о заданном файле. Результат работы приложения показан на рисунке 2.1.

```
parallels@ubuntu-linux-22-04-02-desktop:~$ ./Lab-09a /home/parallels/A.txt
Имя файла: /home/parallels/A.txt
Размер: 760 Б (0.74 КиБ, 0.00 МиБ)
Тип: обычный файл
Время создания: 30.12.2025 22:16
Последний доступ: 30.12.2025 22:16
Последнее изменение: 30.12.2025 22:16

--- Содержимое файла ---
Alexander Ivanov
Dmitry Petrov
Anna Sidorova
Maxim Kuznetsov
```

Рисунок 2.1 – Результат работы Lab-09a

Отличие реализации под Linux по сравнению с Windows обусловлено особенностями операционной системы. В Linux отсутствует встроенный и универсальный механизм, позволяющий надежно определить, является ли файл текстовым или бинарным, поэтому соответствующая функциональность в данной версии приложения не реализовывалась.

2.2 Приложение Lab-09b

В ходе выполнения лабораторной работы было разработано консольное приложение Lab-09b, которое является функциональным аналогом приложения Lab-09b, реализованного для платформы Windows. Логика работы приложения и набор выполняемых операций полностью сохранены, однако реализация адаптирована под особенности операционной системы Linux.

```
0. Выход
> 1
Введите путь к файлу: /home/parallels/A.txt
Файл открыт. Строк: 49, Размер: 760 байт.
```

Рисунок 2.2 – Результат второй программы опции 1

Для выполнения задания создается и компилируется консольное приложение, в которое добавляется код, представленный в листинге Г. После успешной сборки программа запускается, и пользователь последовательно выполняет операции работы с файлом.

```
0. Выход
> 2
Позиция вставки (0 — начало, -1 — конец, >0 — номер): 0
Введите строку: Yan Pavlovich
Строка добавлена на позицию 0.

Меню:
1. Открыть файл
2. Вставить строку
3. Удалить строку
4. Вывести строку
5. Вывести файл
6. Закрыть файл
0. Выход
> 5
1. Yan Pavlovich
2. Alexander Ivanov
3. Dmitry Petrov
4. Anna Sidorova
```

Рисунок 2.3 – Результат второй программы опции 2 и 5

```
0. Выход
> 3
Позиция удаления (0 — первая, -1 — последняя, >0 — номер): 0
Строка удалена с позиции 0.

Меню:
1. Открыть файл
2. Вставить строку
3. Удалить строку
4. Вывести строку
5. Вывести файл
6. Закрыть файл
0. Выход
> 5
1. Alexander Ivanov
2. Dmitry Petrov
3. Anna Sidorova
```

Рисунок 2.4 – Результат второй программы опции 3

```
0. Выход
> 4
Позиция вывода (0 — первая, -1 — последняя, >0 — номер): 0
1: Alexander Ivanov
```

Рисунок 2.5 – Результат второй программы опции 4

Рисунок 2.6 – Результат второй программы опции 6

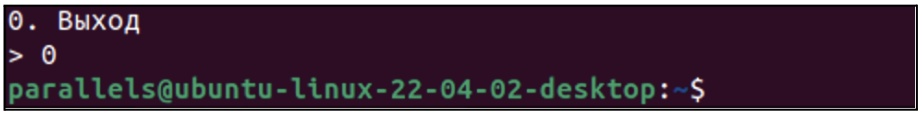

Рисунок 2.7 – Результат второй программы опции 0

На рисунках 2.2-2.7 представлен функционал работы приложения Lab-09c.

Основное отличие реализации под Linux по сравнению с версией для Windows заключается в использовании других системных вызовов и библиотечных функций для работы с файлами и строками, что связано с различиями между POSIX API и WinAPI.

2.3 Приложение Lab-09c

В рамках лабораторной работы было разработано консольное приложение Lab-09c, которое является полным аналогом приложения Lab-09c для платформы Windows, но реализовано с использованием интерфейса inotify. Данное приложение предназначено для мониторинга изменений в файловой системе и поддерживает обработку всех типов событий, предусмотренных POSIX API. Для выполнения задания создается и компилируется консольное приложение, в которое добавляется код, представленный в листинге Е. После запуска программы выполняется ряд действий в отслеживаемой директории, в результате чего в консоли отображаются уведомления о происходящих изменениях. Пример работы приложения приведен на рисунке 2.8.

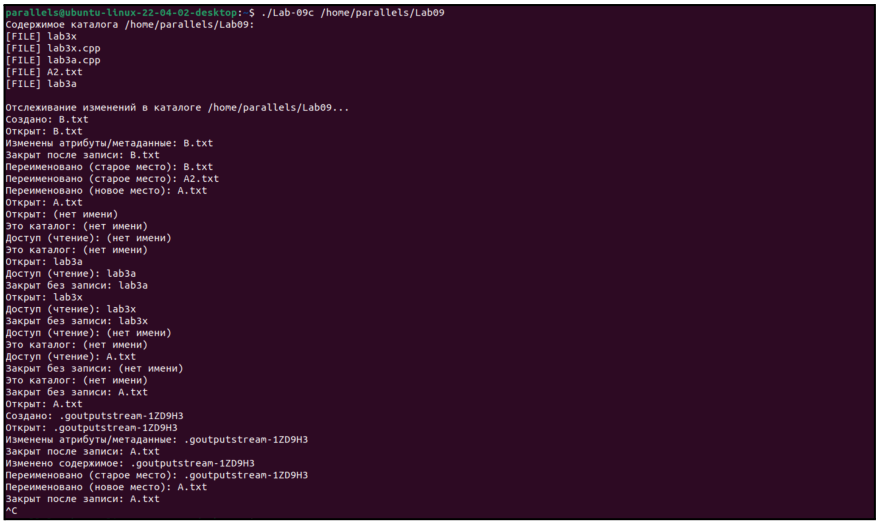

Рисунок 2.8 – Результат работы Lab-09c

Основное различие между реализациями под Linux и Windows заключается в используемых механизмах мониторинга каталогов. В Windows для отслеживания изменений применяется функция ReadDirectoryChangesW, относящаяся к WinAPI, тогда как в Linux используется inotify, представляющий собой системный интерфейс ядра, предназначенный для наблюдения за изменениями файлов и каталогов.

Заключение

В процессе выполнения лабораторной работы № 9 было сформировано более углубленное понимание принципов работы с файлами и каталогами, механизмов асинхронного ввода-вывода, а также выявлены ключевые различия в их реализации в операционных системах Windows и Linux.

На платформе Windows были реализованы приложения Lab-09a, Lab-09b, Lab-09c, демонстрирующие различные способы взаимодействия с файловой системой. В рамках данных приложений выполнялось получение информации о файлах и каталогах с использованием WinAPI, редактирование файлового содержимого, а также мониторинг изменений в директориях посредством функции ReadDirectoryChangesW. Отдельное внимание было уделено асинхронному копированию файлов с применением механизма OVERLAPPED I/O и потоков, создаваемых с помощью CreateThread, что позволило изучить особенности параллельной обработки данных и управления системными ресурсами.

Для платформы Linux были разработаны аналогичные приложения, в которых работа с файлами и каталогами осуществлялась на основе системных вызовов POSIX, таких как open, read, write, stat, opendir и readdir. Мониторинг изменений файловой системы выполнялся с использованием интерфейса inotify, а асинхронное копирование файлов реализовывалось при помощи POSIX Threads (pthread_create, pthread_join) и системных вызовов pread и pwrite. Это позволило закрепить навыки адаптации программных решений под различные операционные системы с учетом различий в системных интерфейсах и модели ввода-вывода.

Проведенный сравнительный анализ показал, что при сохранении общей логики работы приложений различия между Windows и Linux проявляются на уровне реализации. Windows предоставляет более высокоуровневые и тесно интегрированные средства для асинхронного ввода-вывода, тогда как Linux предполагает более явное управление потоками и использование POSIX-интерфейсов. В результате была получена более глубокая оценка архитектурных особенностей обеих операционных систем, а также их влияния на производительность, устойчивость приложений и удобство разработки.

Приложение А – Листинг А.1

```
#include <windows.h>
#include <stdio.h>
#include <locale.h>
BOOL IsTextFile(LPCSTR FileName) {
    HANDLE hFile = CreateFileA(FileName, GENERIC_READ, FILE_SHARE_READ, NULL,
        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == INVALID_HANDLE_VALUE) return FALSE;
    BYTE buffer[512];
    DWORD bytesRead;
    BOOL isText = TRUE;
    if (ReadFile(hFile, buffer, sizeof(buffer), &bytesRead, NULL)) {
        for (DWORD i = 0; i < bytesRead; i++) {
            if (buffer[i] < 9 || (buffer[i] > 13 && buffer[i] < 32)) {
                isText = FALSE;
                break;
            }
        }
    }
    CloseHandle(hFile);
    return isText;
}
void PrintInfo(LPSTR FileName) {
    HANDLE hFile = CreateFileA(FileName, GENERIC_READ, FILE_SHARE_READ, NULL,
        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == INVALID_HANDLE_VALUE) {
        printf("Error: failed to open file %s\n", FileName);
        return;
    }
    BY_HANDLE_FILE_INFORMATION fi;
    if (!GetFileInformationByHandle(hFile, &fi)) {
        printf("Error: failed to get file information\n");
        CloseHandle(hFile);
        return;
    }
    LARGE_INTEGER size;
    size.HighPart = fi.nFileSizeHigh;
    size.LowPart = fi.nFileSizeLow;
    printf("File name: %s\n", FileName);
    printf("File size: %lld bytes (%.2f KiB, %.2f MiB)\n",
        size.QuadPart,
        (double)size.QuadPart / 1024.0,
        (double)size.QuadPart / (1024.0 * 1024.0));
    if (fi.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
        printf("Type: directory\n");
    else
        printf("Type: regular file\n");
    SYSTEMTIME st;
    FileTimeToSystemTime(&fi.ftCreationTime, &st);
    printf("Creation time: %02d.%02d.%04d %02d:%02d\n",
        st.wDay, st.wMonth, st.wYear, st.wHour, st.wMinute);
    FileTimeToSystemTime(&fi.ftLastAccessTime, &st);
    printf("Last access time: %02d.%02d.%04d %02d:%02d\n",
        st.wDay, st.wMonth, st.wYear, st.wHour, st.wMinute);
```

```
    FileTimeToSystemTime(&fi.ftLastWriteTime, &st);
    printf("Last modification time: %02d.%02d.%04d %02d:%02d\n",
        st.wDay, st.wMonth, st.wYear, st.wHour, st.wMinute);
    if (IsTextFile(FileName))
        printf("The file is a text file\n");
    else
        printf("The file is a binary file\n");
    CloseHandle(hFile);
}
void PrintText(LPSTR FileName) {
    if (!IsTextFile(FileName)) {
        printf("File %s is not a text file\n", FileName);
        return;
    }
    HANDLE hFile = CreateFileA(FileName, GENERIC_READ, FILE_SHARE_READ, NULL,
        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == INVALID_HANDLE_VALUE) {
        printf("Error: failed to open file %s\n", FileName);
        return;
    }
    BYTE buffer[512];
    DWORD bytesRead;
    while (ReadFile(hFile, buffer, sizeof(buffer), &bytesRead, NULL) &&
bytesRead > 0) {
        fwrite(buffer, 1, bytesRead, stdout);
    }
    CloseHandle(hFile);
}
int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "C");
    if (argc < 2) {
        printf("Usage: %s <file_name>\n", argv[0]);
        return 1;
    }
    PrintInfo(argv[1]);
    printf("\n--- File content ---\n");
    PrintText(argv[1]);
    return 0;
}
```

Приложение Б – Листинг Б.1

```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>

#define MAX_LINE_LEN 1024

HANDLE g_hFile = INVALID_HANDLE_VALUE;
CHAR* g_fileBuf = NULL;
DWORD  g_fileSize = 0;
CHAR** g_lines = NULL;
DWORD  g_lineCount = 0;

BOOL   LoadFileToBuffer();
BOOL   ParseLines();
BOOL   SaveBufferToFile();
BOOL   RefreshFromDisk();
void   FreeGlobals();
BOOL   EnsureOpen();
BOOL   ValidatePosForInsert(INT pos);
BOOL   ValidatePosForAccess(INT pos, DWORD* outIndex);

BOOL OpenFile(LPSTR filePath) {
    if (g_hFile != INVALID_HANDLE_VALUE) {
        printf("Error: file is already open.\n");
        return FALSE;
    }

    g_hFile = CreateFileA(
        filePath,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL
    );
    if (g_hFile == INVALID_HANDLE_VALUE) {
        printf("Error: file does not exist or cannot be opened: %lu\n",
GetLastError());
        return FALSE;
    }

    if (!LoadFileToBuffer()) {
        CloseHandle(g_hFile); g_hFile = INVALID_HANDLE_VALUE;
        return FALSE;
    }
    if (!ParseLines()) {
        FreeGlobals();
        CloseHandle(g_hFile); g_hFile = INVALID_HANDLE_VALUE;
        return FALSE;
```

```
    }

    printf("File opened. Lines: %lu, Size: %lu bytes.\n",
        (unsigned long)g_lineCount, (unsigned long)g_fileSize);
    return TRUE;
}

BOOL CloseFile() {
    if (g_hFile == INVALID_HANDLE_VALUE) {
        printf("Error: file is not open.\n");
        return FALSE;
    }
    FreeGlobals();
    CloseHandle(g_hFile);
    g_hFile = INVALID_HANDLE_VALUE;
    printf("File closed, resources released.\n");
    return TRUE;
}

void FreeGlobals() {
    if (g_lines) { free(g_lines); g_lines = NULL; }
    if (g_fileBuf) { free(g_fileBuf); g_fileBuf = NULL; }
    g_fileSize = 0;
    g_lineCount = 0;
}

BOOL LoadFileToBuffer() {
    LARGE_INTEGER sz;
    if (!GetFileSizeEx(g_hFile, &sz)) {
        printf("Error getting file size: %lu\n", GetLastError());
        return FALSE;
    }
    if (sz.QuadPart > 0x7FFFFFFF) {
        printf("File is too large for this example.\n");
        return FALSE;
    }
    g_fileSize = (DWORD)sz.QuadPart;

    g_fileBuf = (CHAR*)malloc(g_fileSize + 1);
    if (!g_fileBuf) {
        printf("Memory allocation error.\n");
        return FALSE;
    }

    DWORD read = 0;
    if (g_fileSize == 0) {
        g_fileBuf[0] = '\0';
        return TRUE;
    }

    SetFilePointer(g_hFile, 0, NULL, FILE_BEGIN);
    if (!ReadFile(g_hFile, g_fileBuf, g_fileSize, &read, NULL) || read !=
g_fileSize) {
        printf("File read error: %lu\n", GetLastError());
```

```
        return FALSE;
    }
    g_fileBuf[g_fileSize] = '\0';
    return TRUE;
}

BOOL ParseLines() {
    DWORD count = 0;
    for (DWORD i = 0; i < g_fileSize; i++) {
        if (g_fileBuf[i] == '\n') count++;
    }
    if (g_fileSize > 0 && count == 0) count = 1;

    g_lines = (CHAR**)malloc(sizeof(CHAR*) * (count + 1));
    if (!g_lines) {
        printf("Memory allocation error for line array.\n");
        return FALSE;
    }

    DWORD idx = 0;
    CHAR* p = g_fileBuf;
    CHAR* start = p;

    while (*p) {
        if (*p == '\n') {
            g_lines[idx++] = start;
            if (p > start && *(p - 1) == '\r') *(p - 1) = '\0';
            *p = '\0';
            start = p + 1;
        }
        p++;
    }
    if (start < g_fileBuf + g_fileSize) {
        g_lines[idx++] = start;
    }

    g_lineCount = idx;
    return TRUE;
}

BOOL RefreshFromDisk() {
    if (!LoadFileToBuffer()) return FALSE;
    if (g_lines) { free(g_lines); g_lines = NULL; }
    return ParseLines();
}

BOOL SaveBufferToFile() {
    if (g_hFile == INVALID_HANDLE_VALUE) {
        printf("Error: file is not open.\n");
        return FALSE;
    }
    size_t total = 0;
    for (DWORD i = 0; i < g_lineCount; i++) {
        total += strlen(g_lines[i]) + 2;
```

```
    }
    CHAR* out = (CHAR*)malloc(total + 1);
    if (!out) {
        printf("Memory allocation error.\n");
        return FALSE;
    }

    CHAR* w = out;
    for (DWORD i = 0; i < g_lineCount; i++) {
        size_t len = strlen(g_lines[i]);
        memcpy(w, g_lines[i], len);
        w += len;
        *w++ = '\r';
        *w++ = '\n';
    }
    *w = '\0';

    SetFilePointer(g_hFile, 0, NULL, FILE_BEGIN);
    DWORD written = 0;
    if (!WriteFile(g_hFile, out, (DWORD)(w - out), &written, NULL)) {
        printf("Write error: %lu\n", GetLastError());
        free(out);
        return FALSE;
    }
    SetEndOfFile(g_hFile);

    free(out);
    return RefreshFromDisk();
}

BOOL EnsureOpen() {
    if (g_hFile == INVALID_HANDLE_VALUE) {
        printf("Error: file is not open.\n");
        return FALSE;
    }
    return TRUE;
}

BOOL ValidatePosForInsert(INT pos) {
    if (pos == -1 || pos == 0) return TRUE;
    if (pos > 0 && (DWORD)pos <= g_lineCount + 1) return TRUE;
    return FALSE;
}

BOOL ValidatePosForAccess(INT pos, DWORD* outIndex) {
    if (pos == -1) {
        if (g_lineCount == 0) return FALSE;
        *outIndex = g_lineCount - 1;
        return TRUE;
    }
    if (pos == 0) {
        if (g_lineCount == 0) return FALSE;
        *outIndex = 0;
        return TRUE;
```

```
    }
    if (pos > 0) {
        if ((DWORD)pos > g_lineCount) return FALSE;
        *outIndex = (DWORD)pos - 1;
        return TRUE;
    }
    return FALSE;
}

BOOL AddRow(LPSTR row, INT pos) {
    if (!EnsureOpen()) return FALSE;
    if (row == NULL) { printf("Error: row is NULL.\n"); return FALSE; }
    if (!ValidatePosForInsert(pos)) { printf("Error: invalid insert
position.\n"); return FALSE; }

    DWORD insertIdx = (pos == -1) ? g_lineCount : (pos == 0 ? 0 : (DWORD)pos -
1);

    CHAR** newLines = (CHAR**)malloc(sizeof(CHAR*) * (g_lineCount + 1));
    if (!newLines) { printf("Memory allocation error.\n"); return FALSE; }

    for (DWORD i = 0; i < insertIdx; i++) newLines[i] = g_lines[i];

    size_t len = strlen(row);
    CHAR* copy = (CHAR*)malloc(len + 1);
    if (!copy) { free(newLines); printf("Memory allocation error.\n"); return
FALSE; }
    memcpy(copy, row, len + 1);

    newLines[insertIdx] = copy;

    for (DWORD i = insertIdx; i < g_lineCount; i++) newLines[i + 1] =
g_lines[i];

    free(g_lines);
    g_lines = newLines;
    g_lineCount++;

    if (!SaveBufferToFile()) return FALSE;

    printf("Row added at position %ld.\n", (long)pos);
    return TRUE;
}

BOOL RemRow(INT pos) {
    if (!EnsureOpen()) return FALSE;

    DWORD idx = 0;
    if (!ValidatePosForAccess(pos, &idx)) {
        printf("Error: invalid delete position.\n");
        return FALSE;
    }

    CHAR** newLines = (CHAR**)malloc(sizeof(CHAR*) * (g_lineCount - 1));
```

```
    if (!newLines && g_lineCount > 1) {
        printf("Memory allocation error.\n");
        return FALSE;
    }

    if (g_fileBuf == NULL ||
        (g_lines[idx] < g_fileBuf || g_lines[idx] >= g_fileBuf + g_fileSize))
{
        free(g_lines[idx]);
    }

    for (DWORD i = 0; i < idx; i++) newLines[i] = g_lines[i];
    for (DWORD i = idx + 1; i < g_lineCount; i++) newLines[i - 1] =
g_lines[i];

    free(g_lines);
    g_lines = newLines;
    g_lineCount--;

    if (!SaveBufferToFile()) return FALSE;

    printf("Row removed from position %ld.\n", (long)pos);
    return TRUE;
}

BOOL PrintRow(INT pos) {
    if (!EnsureOpen()) return FALSE;

    DWORD idx = 0;
    if (!ValidatePosForAccess(pos, &idx)) {
        printf("Error: invalid position.\n");
        return FALSE;
    }

    printf("Row [%lu]: %s\n", (unsigned long)(idx + 1), g_lines[idx]);
    return TRUE;
}

BOOL PrintRows() {
    if (!EnsureOpen()) return FALSE;
    for (DWORD i = 0; i < g_lineCount; i++) {
        printf("%lu. %s\n", (unsigned long)(i + 1), g_lines[i]);
    }
    return TRUE;
}

static void FlushStdin() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF) {}
}

int main() {
    SetConsoleOutputCP(65001);
    setlocale(LC_ALL, "C");
```

```
    while (1) {
        printf("\nSelect an operation:\n");
        printf("1. Open file\n");
        printf("2. Insert row\n");
        printf("3. Remove row\n");
        printf("4. Print row\n");
        printf("5. Print file\n");
        printf("6. Close file\n");
        printf("0. Exit\n");
        printf("> ");

        int cmd = -1;
        if (scanf("%d", &cmd) != 1) {
            printf("Command input error.\n");
            FlushStdin();
            continue;
        }
        FlushStdin();

        if (cmd == 0) {
            if (g_hFile != INVALID_HANDLE_VALUE) CloseFile();
            break;
        }

        switch (cmd) {
        case 1: {
            char path[MAX_PATH];
            printf("Enter file path: ");
            if (!fgets(path, sizeof(path), stdin)) { printf("Input error.\n");
break; }
            size_t L = strlen(path);
            if (L && (path[L - 1] == '\n' || path[L - 1] == '\r')) path[L - 1]
= '\0';
            OpenFile(path);
            break;
        }
        case 2: {
            int pos;
            char row[MAX_LINE_LEN];
            printf("Enter insert position (0 = start, -1 = end, >0 = index):
");
            if (scanf("%d", &pos) != 1) { printf("Number input error.\n");
FlushStdin(); break; }
            FlushStdin();
            printf("Enter row text: ");
            if (!fgets(row, sizeof(row), stdin)) { printf("Input error.\n");
break; }
            size_t L = strlen(row);
            if (L && (row[L - 1] == '\n' || row[L - 1] == '\r')) row[L - 1] =
'\0';
            AddRow(row, pos);
            break;
        }
```

```
        case 3: {
            int pos;
            printf("Enter delete position (0 = first, -1 = last, >0 = index):
");
            if (scanf("%d", &pos) != 1) { printf("Number input error.\n");
FlushStdin(); break; }
            FlushStdin();
            RemRow(pos);
            break;
        }
        case 4: {
            int pos;
            printf("Enter print position (0 = first, -1 = last, >0 = index):
");
            if (scanf("%d", &pos) != 1) { printf("Number input error.\n");
FlushStdin(); break; }
            FlushStdin();
            PrintRow(pos);
            break;
        }
        case 5:
            PrintRows();
            break;
        case 6:
            CloseFile();
            break;
        default:
            printf("Unknown command.\n");
        }
    }

    return 0;
}
```

Приложение В – Листинг В.1

```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <stdio.h>
#include <locale.h>
void PrintDirectoryContents(LPCWSTR path) {
    WIN32_FIND_DATAW ffd;
    WCHAR searchPath[MAX_PATH];
    HANDLE hFind;
    wsprintfW(searchPath, L"%s\\*", path);
    hFind = FindFirstFileW(searchPath, &ffd);
    if (hFind == INVALID_HANDLE_VALUE) {
        wprintf(L"Error: failed to read directory %s\n", path);
        return;
    }
    wprintf(L"Contents of directory %s:\n", path);
    do {
        if (wcscmp(ffd.cFileName, L".") == 0 || wcscmp(ffd.cFileName, L"..")
== 0)
            continue;
        if (ffd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY)
            wprintf(L"[DIR]  %s\n", ffd.cFileName);
        else
            wprintf(L"[FILE] %s\n", ffd.cFileName);
    } while (FindNextFileW(hFind, &ffd));
    FindClose(hFind);
}
void WatchDirectory(LPCWSTR path) {
    HANDLE hDir = CreateFileW(
        path,
        FILE_LIST_DIRECTORY,
        FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE,
        NULL,
        OPEN_EXISTING,
        FILE_FLAG_BACKUP_SEMANTICS,
        NULL
    );
    if (hDir == INVALID_HANDLE_VALUE) {
        wprintf(L"Error: failed to open directory %s\n", path);
        return;
    }
    BYTE buffer[1024];
    DWORD bytesReturned;
    FILE_NOTIFY_INFORMATION* fni;
    WCHAR fileName[MAX_PATH];
    wprintf(L"\nMonitoring changes in directory %s...\n", path);
    while (1) {
        if (ReadDirectoryChangesW(
            hDir,
            buffer,
            sizeof(buffer),
            FALSE, // first level only
            FILE_NOTIFY_CHANGE_FILE_NAME |
            FILE_NOTIFY_CHANGE_DIR_NAME |
```

```
            FILE_NOTIFY_CHANGE_ATTRIBUTES |
            FILE_NOTIFY_CHANGE_SIZE |
            FILE_NOTIFY_CHANGE_LAST_WRITE |
            FILE_NOTIFY_CHANGE_CREATION |
            FILE_NOTIFY_CHANGE_SECURITY,
            &bytesReturned,
            NULL,
            NULL
        )) {
            fni = (FILE_NOTIFY_INFORMATION*)buffer;
            int len = fni->FileNameLength / sizeof(WCHAR);
            wcsncpy(fileName, fni->FileName, len);
            fileName[len] = L'\0';
            switch (fni->Action) {
            case FILE_ACTION_ADDED:
                wprintf(L"Added: %s\n", fileName); break;
            case FILE_ACTION_REMOVED:
                wprintf(L"Removed: %s\n", fileName); break;
            case FILE_ACTION_MODIFIED:
                wprintf(L"Modified: %s\n", fileName); break;
            case FILE_ACTION_RENAMED_OLD_NAME:
                wprintf(L"Renamed (old name): %s\n", fileName); break;
            case FILE_ACTION_RENAMED_NEW_NAME:
                wprintf(L"Renamed (new name): %s\n", fileName); break;
            default:
                wprintf(L"Event %d for %s\n", fni->Action, fileName);
            }
        }
        else {
            wprintf(L"ReadDirectoryChangesW error: %lu\n", GetLastError());
            break;
        }
    }
    CloseHandle(hDir);
}
int wmain(int argc, wchar_t* argv[]) {
    setlocale(LC_ALL, "ru");
    if (argc < 2) {
        wprintf(L"Usage: %s <directory_path>\n", argv[0]);
        return 1;
    }
    DWORD attrs = GetFileAttributesW(argv[1]);
    if (attrs == INVALID_FILE_ATTRIBUTES || !(attrs &
FILE_ATTRIBUTE_DIRECTORY)) {
        wprintf(L"Error: directory %s does not exist.\n", argv[1]);
        return 1;
    }
    PrintDirectoryContents(argv[1]);
    WatchDirectory(argv[1]);
    return 0;
}
```

Приложение Г – Листинг Г.1

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

void PrintInfo(const char* fileName) {
    struct stat st;
    if (stat(fileName, &st) == -1) {
        perror("Ошибка: не удалось получить информацию о файле");
        return;
    }

    printf("Имя файла: %s\n", fileName);
    printf("Размер: %lld Б (%.2f КиБ, %.2f МиБ)\n",
           (long long)st.st_size,
           (double)st.st_size / 1024.0,
           (double)st.st_size / (1024.0 * 1024.0));

    if (S_ISDIR(st.st_mode))
        printf("Тип: каталог\n");
    else if (S_ISREG(st.st_mode))
        printf("Тип: обычный файл\n");
    else
        printf("Тип: другой объект\n");

    char buf[64];
    struct tm *tm_info;

    tm_info = localtime(&st.st_ctime);
    strftime(buf, sizeof(buf), "%d.%m.%Y %H:%M", tm_info);
    printf("Время создания: %s\n", buf);

    tm_info = localtime(&st.st_atime);
    strftime(buf, sizeof(buf), "%d.%m.%Y %H:%M", tm_info);
    printf("Последний доступ: %s\n", buf);

    tm_info = localtime(&st.st_mtime);
    strftime(buf, sizeof(buf), "%d.%m.%Y %H:%M", tm_info);
    printf("Последнее изменение: %s\n", buf);
}

void PrintText(const char* fileName) {
    int fd = open(fileName, O_RDONLY);
    if (fd == -1) {
```

```
        perror("Ошибка: не удалось открыть файл");
        return;
    }

    char buffer[512];
    ssize_t bytesRead;
    while ((bytesRead = read(fd, buffer, sizeof(buffer))) > 0) {
        fwrite(buffer, 1, bytesRead, stdout);
    }

    close(fd);
}

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "ru_RU.UTF-8");

    if (argc < 2) {
        printf("Использование: %s <имя_файла>\n", argv[0]);
        return 1;
    }

    PrintInfo(argv[1]);
    printf("\n--- Содержимое файла ---\n");
    PrintText(argv[1]);

    return 0;
}
```

Приложение Д – Листинг Д.1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <locale.h>

#define MAX_LINE_LEN 1024
int g_fd = -1;
char *g_fileBuf = NULL;
size_t g_fileSize = 0;
char **g_lines = NULL;
size_t g_lineCount = 0;
static char* dup_n(const char* src, size_t n) {
    char* s = (char*)malloc(n + 1);
    if (!s) return NULL;
    memcpy(s, src, n);
    s[n] = '\0';
    return s;
}
void FreeGlobals() {
    if (g_lines) {
        for (size_t i = 0; i < g_lineCount; i++) {
            free(g_lines[i]);
        }
        free(g_lines);
        g_lines = NULL;
    }
    if (g_fileBuf) { free(g_fileBuf); g_fileBuf = NULL; }
    g_fileSize = 0;
    g_lineCount = 0;
}

int EnsureOpen() {
    if (g_fd == -1) {
        printf("Ошибка: файл не открыт.\n");
        return 0;
    }
    return 1;
}

int LoadFileToBuffer() {

    struct stat st;
    if (fstat(g_fd, &st) == -1) {
        perror("Ошибка получения размера файла");
        return 0;
```

```
    }
    g_fileSize = st.st_size;
    g_fileBuf = (char*)malloc(g_fileSize + 1);
    if (!g_fileBuf) {
        printf("Ошибка выделения памяти.\n");
        return 0;
    }
    if (g_fileSize == 0) { g_fileBuf[0] = '\0'; return 1; }

    if (lseek(g_fd, 0, SEEK_SET) < 0) {
        perror("Ошибка позиционирования файла");
        return 0;
    }
    ssize_t r = read(g_fd, g_fileBuf, g_fileSize);
    if (r < 0 || (size_t)r != g_fileSize) {
        perror("Ошибка чтения файла");
        return 0;
    }
    g_fileBuf[g_fileSize] = '\0';
    return 1;
}

int ParseLines() {

    size_t count = 0;
    for (size_t i = 0; i < g_fileSize; i++) {
        if (g_fileBuf[i] == '\n') count++;
    }
    if (g_fileSize > 0 && count == 0) count = 1;

    g_lines = (char**)malloc(sizeof(char*) * (count ? count : 1));
    if (!g_lines) { printf("Ошибка выделения памяти.\n"); return 0; }

    size_t idx = 0;
    char* start = g_fileBuf;
    for (size_t i = 0; i < g_fileSize; i++) {
        if (g_fileBuf[i] == '\n') {
            // строка от start до i-1, возможно CRLF — убираем '\r'
            size_t len = (size_t)(g_fileBuf + i - start);
            if (len > 0 && start[len - 1] == '\r') len--;
            char* s = dup_n(start, len);
            if (!s) { printf("Ошибка выделения памяти строки.\n"); return 0; }
            g_lines[idx++] = s;
            start = g_fileBuf + i + 1;

        }
    }
    if ((size_t)(start - g_fileBuf) < g_fileSize) {
        size_t len = (size_t)(g_fileBuf + g_fileSize - start);
```

```
        if (len > 0) {
            if (len > 0 && start[len - 1] == '\r') len--;
            char* s = dup_n(start, len);
            if (!s) { printf("Ошибка выделения памяти строки.\n"); return 0; }
            g_lines[idx++] = s;
        }
    }

    g_lineCount = idx;
    return 1;
}

int RefreshFromDisk() {
    FreeGlobals();
    if (!LoadFileToBuffer()) return 0;
    return ParseLines();
}

int SaveBufferToFile() {
    if (!EnsureOpen()) return 0;
    size_t total = 0;
    for (size_t i = 0; i < g_lineCount; i++) total += strlen(g_lines[i]) + 1;

    char *out = (char*)malloc(total + 1);
    if (!out) { printf("Ошибка выделения памяти.\n"); return 0; }

    char *w = out;
    for (size_t i = 0; i < g_lineCount; i++) {
        size_t len = strlen(g_lines[i]);
        memcpy(w, g_lines[i], len);
        w += len;
        *w++ = '\n';
    }
    *w = '\0';

    if (lseek(g_fd, 0, SEEK_SET) < 0) {
        perror("Ошибка позиционирования файла");
        free(out);
        return 0;
    }
    ssize_t wr = write(g_fd, out, (size_t)(w - out));
    if (wr < 0 || (size_t)wr != (size_t)(w - out)) {
        perror("Ошибка записи");
        free(out);

        return 0;
    }
    if (ftruncate(g_fd, (off_t)(w - out)) < 0) {
        perror("Ошибка обрезки файла");
```

```
        free(out);
        return 0;
    }
    free(out);
    return RefreshFromDisk();
}

int OpenFile(const char *path) {
    if (g_fd != -1) {
        printf("Ошибка: файл уже открыт.\n");
        return 0;
    }
    g_fd = open(path, O_RDWR);
    if (g_fd == -1) {
        perror("Ошибка открытия файла");
        return 0;
    }
    if (!LoadFileToBuffer() || !ParseLines()) {
        close(g_fd); g_fd = -1; FreeGlobals();
        return 0;
    }
    printf("Файл открыт. Строк: %lu, Размер: %lu байт.\n",
           (unsigned long)g_lineCount, (unsigned long)g_fileSize);
    return 1;
}

int CloseFile() {
    if (g_fd == -1) {
        printf("Ошибка: файл не открыт.\n");
        return 0;
    }
    FreeGlobals();
    close(g_fd);
    g_fd = -1;
    printf("Файл закрыт.\n");
    return 1;
}

int AddRow(const char *row, int pos) {
    if (!EnsureOpen()) return 0;
    if (!row) { printf("Ошибка: строка NULL.\n"); return 0; }

    size_t insertIdx = 0;
    if (pos == -1) insertIdx = g_lineCount;
    else if (pos == 0) insertIdx = 0;

       else if (pos > 0 && (size_t)pos <= g_lineCount + 1) insertIdx =
(size_t)pos - 1;
    else { printf("Ошибка: недопустимая позиция.\n"); return 0; }
```

```
    char **newLines = (char**)malloc(sizeof(char*) * (g_lineCount + 1));
    if (!newLines) { printf("Ошибка выделения памяти.\n"); return 0; }

    for (size_t i = 0; i < insertIdx; i++) newLines[i] = g_lines[i];

    char *copy = strdup(row);
     if (!copy) { free(newLines); printf("Ошибка выделения памяти строки.\n");
return 0; }
    newLines[insertIdx] = copy;

      for (size_t i = insertIdx; i < g_lineCount; i++) newLines[i + 1] =
g_lines[i];

    free(g_lines);
    g_lines = newLines;
    g_lineCount++;

    if (!SaveBufferToFile()) return 0;
    printf("Строка добавлена на позицию %d.\n", pos);
    return 1;
}

int RemRow(int pos) {
    if (!EnsureOpen()) return 0;
    if (g_lineCount == 0) { printf("Ошибка: файл пуст.\n"); return 0; }

    size_t idx;
    if (pos == -1) idx = g_lineCount - 1;
    else if (pos == 0) idx = 0;
    else if (pos > 0 && (size_t)pos <= g_lineCount) idx = (size_t)pos - 1;
    else { printf("Ошибка: недопустимая позиция.\n"); return 0; }

    char **newLines = NULL;
    if (g_lineCount > 1) {
        newLines = (char**)malloc(sizeof(char*) * (g_lineCount - 1));
        if (!newLines) { printf("Ошибка выделения памяти.\n"); return 0; }
    }

    free(g_lines[idx]);

    for (size_t i = 0; i < idx; i++) if (newLines) newLines[i] = g_lines[i];
     for (size_t i = idx + 1; i < g_lineCount; i++) if (newLines) newLines[i -
1] = g_lines[i];

    free(g_lines);
    g_lines = newLines;
    g_lineCount--;
```

```
    if (!SaveBufferToFile()) return 0;
    printf("Строка удалена с позиции %d.\n", pos);
    return 1;
}

int PrintRow(int pos) {
    if (!EnsureOpen()) return 0;
    if (g_lineCount == 0) { printf("Ошибка: файл пуст.\n"); return 0; }

    size_t idx;
    if (pos == -1) idx = g_lineCount - 1;
    else if (pos == 0) idx = 0;
    else if (pos > 0 && (size_t)pos <= g_lineCount) idx = (size_t)pos - 1;
    else { printf("Ошибка: недопустимая позиция.\n"); return 0; }

    printf("%lu: %s\n", (unsigned long)(idx + 1), g_lines[idx]);
    return 1;
}

int PrintRows() {
    if (!EnsureOpen()) return 0;
    for (size_t i = 0; i < g_lineCount; i++) {
        printf("%lu. %s\n", (unsigned long)(i + 1), g_lines[i]);
    }
    return 1;
}

int main() {
    setlocale(LC_ALL, "ru_RU.UTF-8");
    while (1) {
        printf("\nМеню:\n");
        printf("1. Открыть файл\n");
        printf("2. Вставить строку\n");
        printf("3. Удалить строку\n");
        printf("4. Вывести строку\n");
        printf("5. Вывести файл\n");
        printf("6. Закрыть файл\n");
        printf("0. Выход\n> ");

        int cmd;
        if (scanf("%d", &cmd) != 1) { printf("Ошибка ввода.\n"); break; }
        getchar(); // очистка \n

        if (cmd == 0) {
            if (g_fd != -1) CloseFile();
            break;
        }

        switch (cmd) {
```

```
        case 1: {
            char path[256];
            printf("Введите путь к файлу: ");
                    if (!fgets(path, sizeof(path), stdin)) { printf("Ошибка
ввода.\n"); break; }
            path[strcspn(path, "\n")] = '\0';
            OpenFile(path);
            break;
        }
        case 2: {
            char row[MAX_LINE_LEN];
            int pos;
            printf("Позиция вставки (0 — начало, -1 — конец, >0 — номер): ");
            if (scanf("%d", &pos) != 1) { printf("Ошибка ввода.\n"); break; }
            getchar();
            printf("Введите строку: ");
             if (!fgets(row, sizeof(row), stdin)) { printf("Ошибка ввода.\н");
break; }
            row[strcspn(row, "\n")] = '\0';
            AddRow(row, pos);
            break;
        }
        case 3: {
            int pos;
            printf("Позиция удаления (0 — первая, -1 — последняя, >0 — номер):
");
            if (scanf("%d", &pos) != 1) { printf("Ошибка ввода.\n"); break; }
            getchar();
            RemRow(pos);
            break;
        }
        case 4: {
            int pos;
             printf("Позиция вывода (0 — первая, -1 — последняя, >0 — номер):
");
            if (scanf("%d", &pos) != 1) { printf("Ошибка ввода.\n"); break; }
            getchar();
            PrintRow(pos);

            break;
        }
        case 5:
            PrintRows();
            break;
        case 6:
            CloseFile();
            break;
        default:
            printf("Неизвестная команда.\n");
```

```
        }
    }
    return 0;
}
```

Приложение Е – Листинг Е.1

```
#define _XOPEN_SOURCE 700
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/inotify.h>
#include <sys/stat.h>
#include <dirent.h>
#include <limits.h>

#ifndef BUF_LEN
#define BUF_LEN (1024 * (sizeof(struct inotify_event) + NAME_MAX + 1))
#endif

static void PrintDirectoryContents(const char* path) {
    DIR* d = opendir(path);
    if (!d) {
        fprintf(stderr, "Ошибка: не удалось прочитать каталог %s: %s\n", path,
strerror(errno));
        return;
    }
    printf("Содержимое каталога %s:\n", path);
    struct dirent* de;
    while ((de = readdir(d)) != NULL) {
          if (strcmp(de->d_name, ".") == 0 || strcmp(de->d_name, "..") == 0)
continue;
        if (de->d_type == DT_DIR) printf("[DIR]  %s\n", de->d_name);
        else if (de->d_type == DT_REG) printf("[FILE] %s\n", de->d_name);
        else printf("[OTHER] %s\n", de->d_name);
    }
    closedir(d);
}

static void print_mask(uint32_t mask, const char* name) {
    if (mask & IN_CREATE)          printf("Создано: %s\n", name);

    if (mask & IN_DELETE)          printf("Удалено: %s\n", name);
    if (mask & IN_MODIFY)          printf("Изменено содержимое: %s\n", name);
     if (mask & IN_ATTRIB)             printf("Изменены атрибуты/метаданные:
%s\n", name);
     if (mask & IN_MOVED_FROM)        printf("Переименовано (старое место):
%s\n", name);
    if (mask & IN_MOVED_TO)        printf("Переименовано (новое место): %s\n",
name);

    if (mask & IN_DELETE_SELF)     printf("Сам объект наблюдения удалён\n");
```

```
        if (mask & IN_MOVE_SELF)             printf("Сам объект наблюдения
перемещён\n");
    if (mask & IN_ACCESS)          printf("Доступ (чтение): %s\n", name);
    if (mask & IN_OPEN)            printf("Открыт: %s\n", name);
    if (mask & IN_CLOSE_WRITE)     printf("Закрыт после записи: %s\n", name);
    if (mask & IN_CLOSE_NOWRITE)   printf("Закрыт без записи: %s\n", name);

       if (mask & IN_IGNORED)              printf("Наблюдение снято ядром
(IGNORED)\n");
        if (mask & IN_Q_OVERFLOW)          printf("Потеря событий (очередь
переполнена)\n");
    if (mask & IN_UNMOUNT)         printf("ФС размонтирована\n");
    if (mask & IN_ISDIR)           printf("Это каталог: %s\n", name);
    if (mask & IN_ONLYDIR)         printf("Событие только для каталогов\n");
    if (mask & IN_EXCL_UNLINK)     printf("Исключить объекты после unlink\n");
}

static int is_directory(const char* path) {
    struct stat st;
    if (stat(path, &st) != 0) return 0;
    return S_ISDIR(st.st_mode);
}

static void WatchDirectory(const char* path, int recursive) {
    int fd = inotify_init1(IN_NONBLOCK);
    if (fd == -1) {
        fprintf(stderr, "Ошибка inotify_init1: %s\n", strerror(errno));
        return;
    }

    uint32_t mask =
        IN_CREATE | IN_DELETE | IN_MODIFY | IN_ATTRIB |
        IN_MOVED_FROM | IN_MOVED_TO |
        IN_DELETE_SELF | IN_MOVE_SELF |
        IN_ACCESS | IN_OPEN | IN_CLOSE_WRITE | IN_CLOSE_NOWRITE |
           IN_IGNORED | IN_Q_OVERFLOW | IN_UNMOUNT | IN_ISDIR | IN_ONLYDIR |
IN_EXCL_UNLINK;

    int wd = inotify_add_watch(fd, path, mask);
    if (wd == -1) {
              fprintf(stderr, "Ошибка inotify_add_watch для %s: %s\n", path,
strerror(errno));
        close(fd);
        return;
    }

    if (recursive) {
        DIR* d = opendir(path);
        if (d) {
```

```
            struct dirent* de;
            char subpath[PATH_MAX];
            while ((de = readdir(d)) != NULL) {
                  if (strcmp(de->d_name, ".") == 0 || strcmp(de->d_name, "..")
== 0) continue;
                snprintf(subpath, sizeof(subpath), "%s/%s", path, de->d_name);
                if (is_directory(subpath)) {
                    int swd = inotify_add_watch(fd, subpath, mask);
                    if (swd == -1) {
                              fprintf(stderr, "Ошибка add_watch для %s: %s\n",
subpath, strerror(errno));
                    } else {
                         printf("Добавлено наблюдение: %s (wd=%d)\n", subpath,
swd);
                    }
                }
            }
            closedir(d);
        }
    }

    printf("\nОтслеживание изменений в каталоге %s...\n", path);

    char buf[BUF_LEN];
    for (;;) {
        ssize_t len = read(fd, buf, sizeof(buf));
        if (len == -1) {
            if (errno == EAGAIN) { // нет данных — чуть подождать

                usleep(50 * 1000);
                continue;
            } else {
                              fprintf(stderr, "Ошибка чтения событий: %s\n",
strerror(errno));
                break;
            }
        }
        if (len == 0) continue;

        for (char *ptr = buf; ptr < buf + len; ) {
            struct inotify_event *ev = (struct inotify_event *)ptr;
            const char* name = (ev->len > 0 && ev->name[0]) ? ev->name : "(нет
имени)";
            print_mask(ev->mask, name);

             if (recursive && (ev->mask & IN_CREATE) && (ev->mask & IN_ISDIR))
{
                char newdir[PATH_MAX];
                snprintf(newdir, sizeof(newdir), "%s/%s", path, name);
                int swd = inotify_add_watch(fd, newdir, mask);
```

```
                if (swd != -1) {
                            printf("Добавлено наблюдение на новый каталог: %s
(wd=%d)\n", newdir, swd);
                }
            }

            ptr += sizeof(struct inotify_event) + ev->len;
        }
    }

    close(fd);
}

int main(int argc, char* argv[]) {
    setlocale(LC_ALL, "ru_RU.UTF-8");

    if (argc < 2) {
              printf("Использование: %s <путь_к_каталогу> [--recursive]\n",
argv[0]);
        return 1;
    }

    const char* path = argv[1];
    int recursive = (argc >= 3 && strcmp(argv[2], "--recursive") == 0);

    struct stat st;
    if (stat(path, &st) != 0 || !S_ISDIR(st.st_mode)) {

        fprintf(stderr, "Ошибка: каталог %s не существует.\n", path);
        return 1;
    }

    PrintDirectoryContents(path);
    WatchDirectory(path, recursive);
    return 0;
}
```