

Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

Практическая работа №4 по дисциплине
«Системное программирование»
Тема работы: «Файловый ввод-вывод»

Студент гр. БИБ 201
Морин Денис Александрович
«28» января 2023 г.

Руководитель
Преподаватель В.И.Морозов
«__» _____ 2023 г.

Москва 2023

Оглавление

1 Цели и задачи практической работы.....	3
2 Ход работы.....	4
3 Вывод.....	12
Приложение А.....	13
Приложение Б.....	16
Приложение В.....	18

1 Цель работы.

Целью данной работы является знакомство с файловым вводом-выводом в операционных системах Windows и Linux и в качестве закрепления материала написание программы, которая должна выполнять условия, которые находятся в Приложении В.

2 Ход работы

В начале файла main.c программы для Windows подключаются библиотеки для их дальнейшего использования и определяются константы (Листинг 1).

```
#include "stdio.h"
#include "windows.h"

#define ERROR_CODE 1
#define SUCCESS 0

#define SIZE 128
```

Листинг 1 – Подключение библиотек и введение констант в main.c

Для вывода в консоль сообщения в дальнейшем будет передаваться строка для вывода в функцию prnt, которая и будет выполнять вышеуказанное действие (Листинг 2).

```
int prnt(LPSTR message){

    // Извлекает дескриптор стандартного устройства вывода
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

    // Если не вернул дескриптор = ошибка
    if (hConsole == INVALID_HANDLE_VALUE){
        return ERROR_CODE;
    }

    // Записывает символьную строку в буфер экрана консоли
    // dwBytesWritten - Указатель на переменную, которая
    // получает количество фактически записанных символов.
    if(!WriteConsole(hConsole, message,
                    strlen(message), 0, NULL)){
        return ERROR_CODE;
    }

    return SUCCESS;
}
```

Листинг 2 – Функция для печати текста в консоль

В главной функции мы передаем аргументы для их счета и считывания значений. Имеются проверки на число переданных аргументов (их строго 4). Из массива с аргументами передаются число и название существующего файла. Делается проверка на корректность числа (Листинг 3).

```
// argv["1abc", "2"]
int main(int argc, char ** argv){

    if (argc != 3){
        LPSTR message = "It is needed 3 arguments";
        prnt(message);
        return ERROR_CODE;
    }

    int N = atoi(argv[1]);
    DWORD dwBytesWritten = 0;

    if (N < 0){
        LPSTR message = "N must be a positive number";
        prnt(message);
        return ERROR_CODE;
    }
}
```

Листинг 3 – Функция main файла main.c

Создается дескриптор файла, который уже существует, открывается файл для дальнейшей записи. Если ошибка при создании дескриптора, то данный случай обрабатывается, завершая программу. (Листинг 4).

```
HANDLE hFile = CreateFile(argv[2], GENERIC_WRITE,
                          0, NULL,
                          OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
                          NULL);

if (hFile == INVALID_HANDLE_VALUE){
    LPSTR message = "Error occurred! File is not existed";
    prnt(message);
    return ERROR_CODE;
}
```

Листинг 4 – Создание дескриптора файла

Выполняется условие задания (Листинг 5).

```
if (N <= len1) {  
    argv[3][N] = '\\0';  
}
```

Листинг 5 – Условие задания

Запрашивается строка на ввод в консоль, выделяется под нее память, записывается строка. Также вычисляется длина введенной строки (Листинг 6).

```
printf("%s%d%s\\n", "Input string to ", SIZE, " chars");  
HANDLE hin = GetStdHandle(STD_INPUT_HANDLE);  
DWORD size = SIZE*sizeof(CHAR);  
LPSTR buf = malloc(SIZE*sizeof(CHAR));  
int len;  
  
if (ReadFile(hin, buf, size, 0, NULL)) {  
    // Переменная, в которую будет занесен адрес первой найденной  
    строки  
    char *istr;  
    istr = strstr(buf, "\\n");  
    len = istr-buf-1;  
    CloseHandle(hin);  
}  
else {  
    LPSTR message = "Read error: cannot read from console";  
    prnt(message);  
    return ERROR_CODE;  
}
```

Листинг 6 – Запись строки из консоли

Выделяется память для дополнительного задания с помощью HeapAlloc. Записывается текст в файл, иначе возникает ошибка. Закрывается дескриптор файла и освобождается память, которая занималась ранее (Листинг 7).

```
// GetProcessHeap() - Извлекает дескриптор кучи вызывающего  
процесса
```

```

        // Выделенная память будет инициализирована до нуля
        LPVOID pBuf = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
strlen(buf)*sizeof(char));
        if (!pBuf){
            LPSTR message = "HeapAlloc failed";
            prnt(message);
        }

        pBuf = buf;
        free(buf);

        // Записывает данные в указанный файл
        // Указатель на структуру с параметрами асинхронной записи
        if (!WriteFile( hFile, pBuf, strlen(pBuf),
                        &dwBytesWritten , NULL)) {
            LPSTR message = "Error occurred! Data is not written";
            prnt(message);
            return ERROR_CODE;
        }

        // Закрывает дескриптор открытого объекта
        CloseHandle(hFile);

        // Освобождает блок памяти, выделенный из кучи
        HeapFree(GetProcessHeap(), 0, pBuf);

        return SUCCESS;
    }

```

Листинг 7 – Продолжение функции main

Далее рассматривается аналогичная программа, подстроенная под операционную систему Linux.

Импортируются библиотеки для использования winapi функций, для работы со строками. Определяются константные переменные (Листинг 8).

```

#include "fcntl.h"
#include "stdlib.h"
#include "string.h"
#include "unistd.h"

#define ERROR_CODE 1

```

```
#define SUCCESS 0
#define SIZE 128
```

Листинг 8 – Импорт библиотек и создание констант в файле `ari.c`

Главная функция `ari.c` подобна главной функции на `main.c`, за исключением передачи аргументов в функциях создания файла, записи в него, а также работы с функциями, которые работают с памятью (Листинг 9).

```
int main(int argc, char *argv[]){

    if (argc < 3){
        char *message = "It is needed 3 arguments\n";
        write(STDOUT_FILENO, message, strlen(message));
        return ERROR_CODE;
    }

    if (argc > 3){
        char *message = "Expected 3 arguments\nLast argument
should be in \"\"\n";
        write(STDOUT_FILENO, message, strlen(message));
        return ERROR_CODE;
    }

    char *buf = malloc(SIZE*sizeof(char));
    char *s = "Input string to 128 chars\n";
    write(STDOUT_FILENO, s, strlen(s));
    int len = read(STDIN_FILENO, buf, SIZE)-1;

    if (len == 0){
        char *message = "Read error: cannot read from console";
        write(STDOUT_FILENO, message, strlen(message));
        return ERROR_CODE;
    }

    int N = atoi(argv[1]);

    if (N < 0){
        char *message = "N must be a positive number";
        write(STDOUT_FILENO, message, strlen(message));
        return ERROR_CODE;
    }
}
```



```

    }

    int file = open(argv[2], O_WRONLY);

    if (file < 0){
        char *message = "Error occurred! File is not existed\n";
        write(STDOUT_FILENO, message, strlen(message));
        return ERROR_CODE;
    }

    if (N <= len){
        buf[N] = '\0';
    }

    if (!write(file, buf, strlen(buf))){
        char *message = "Error occurred! Data is not written\n";
        write(STDOUT_FILENO, message, strlen(message));
        return ERROR_CODE;
    }

    free(buf);

    close(file);

    return SUCCESS;
}

```

Листинг 9 – Функция main файла api.c

В режиме отладки программы main.c на рисунке 1 представлены изначально заданные аргументы.

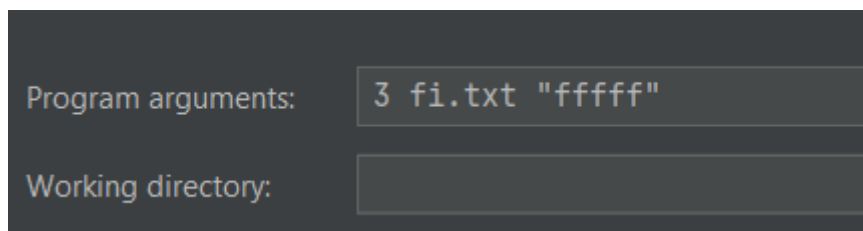
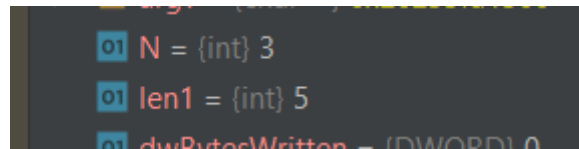


Рисунок 1 – Аргументы для запуска программы в режиме отладки

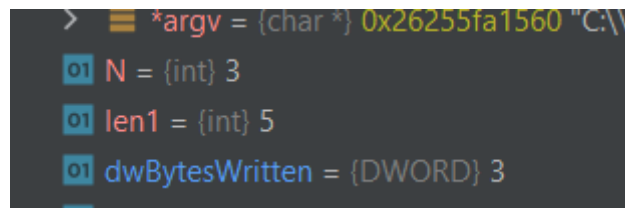
Далее видны значения переменной N и длины переданного текста (рисунок 2).



```
01 N = {int} 3
01 len1 = {int} 5
01 dwBytesWritten = {DWORD} 0
```

Рисунок 2 – Значения переменных N и len1

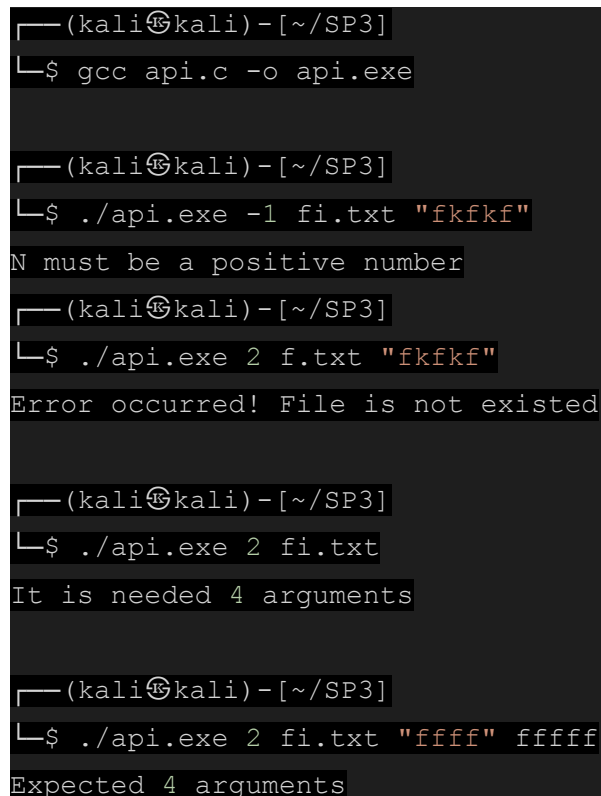
Переменная dwBytesWritten равна 3 после записи в файл, то есть записалось 3 байта, чему и равно N (рисунок 3).



```
> == *argv = {char *} 0x26255fa1560 "C:\\\\
01 N = {int} 3
01 len1 = {int} 5
01 dwBytesWritten = {DWORD} 3
```

Рисунок 3 – Значение переменной dwBytesWritten

Компилирование файла, всевозможные ошибки (кроме выделения памяти из блока кучи) на Linux и правильный запуск программы можно посмотреть в листинге 10.



```
(kali㉿kali)-[~/SP3]
└─$ gcc api.c -o api.exe

(kali㉿kali)-[~/SP3]
└─$ ./api.exe -1 fi.txt "fkfkf"
N must be a positive number

(kali㉿kali)-[~/SP3]
└─$ ./api.exe 2 f.txt "fkfkf"
Error occurred! File is not existed

(kali㉿kali)-[~/SP3]
└─$ ./api.exe 2 fi.txt
It is needed 4 arguments

(kali㉿kali)-[~/SP3]
└─$ ./api.exe 2 fi.txt "ffff" fffff
Expected 4 arguments
```

```

Last argument should be in ""

(kali㉿kali)-[~/SP3]
└─$ ./api.exe 2 fi.txt ""
Error occurred! Data is not written

(kali㉿kali)-[~/SP3]
└─$ ./api.exe 2 fi.txt "ffff fffff"

(kali㉿kali)-[~/SP3]
└─$

```

Листинг 10 – Компиляция и запуск программы несколько раз с разными аргументами

На рисунке 4 показано, что в существующий файл успешно записан текст.



Рисунок 4 – Содержимое файла fi.txt

3 Вывод

На этой практике были изучены API операционных систем, файловый ввод-вывод на разных операционных системах.

Исходный код программы `main.c` для Windows и `api.c` для Linux находятся в приложении А и Б соответственно.

ПРИЛОЖЕНИЕ А.

Программа для операционной системы Windows

```
#include "stdio.h"
#include "windows.h"

#define ERROR_CODE 1
#define SUCCESS 0

#define SIZE 128

int prnt(LPSTR message){

    // Извлекает дескриптор стандартного устройства вывода
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

    // Если не вернул дескриптор = ошибка
    if (hConsole == INVALID_HANDLE_VALUE){
        return ERROR_CODE;
    }

    // Записывает символьную строку в буфер экрана консоли
    // dwBytesWritten - Указатель на переменную, которая
    // получает количество фактически записанных символов.
    if(!WriteConsole(hConsole, message,
                    strlen(message), 0, NULL)){
        return ERROR_CODE;
    }

    return SUCCESS;
}

// argv["1abc", "2"]
int main(int argc, char ** argv){

    if (argc < 3){
        LPSTR message = "It is needed 3 arguments";
        prnt(message);
        return ERROR_CODE;
    }

    if (argc > 3){
```

```

        LPSTR message = "Expected 3 arguments\nLast argument should be
in \"\"";
        prnt(message);
        return ERROR_CODE;
    }

    int N = atoi(argv[1]);
    DWORD dwBytesWritten = 0;

    if (N < 0){
        LPSTR message = "N must be a positive number";
        prnt(message);
        return ERROR_CODE;
    }

    printf("%s%d%s\n", "Input string to ", SIZE, " chars");
    HANDLE hin = GetStdHandle(STD_INPUT_HANDLE);
    DWORD size = SIZE*sizeof(CHAR);
    LPSTR buf = malloc(SIZE*sizeof(CHAR));
    int len;

    if (ReadFile(hin, buf, size, 0, NULL)){
        // Переменная, в которую будет занесен адрес первой найденной
строки
        char *istr;
        istr = strstr(buf, "\n");
        len = istr-buf-1;
        CloseHandle(hin);
    }
    else{
        LPSTR message = "Read error: cannot read from console";
        prnt(message);
        return ERROR_CODE;
    }

    // открывает файл
    /*
        имя файла,
        флаг доступа на запись
        совместный доступ к файлу запрещён
        указатель на структуру с параметрами безопасности для

```

```

вновь создаваемого
файла
    производится открытие существующего файла
    дополнительные атрибуты для вновь создаваемого файла
    дескриптор файла-шаблона, из которого берутся значения для
    флагов вновь
создаваемого файла
    */
    HANDLE hFile = CreateFile(argv[2], GENERIC_WRITE,
                                0, NULL,
                                OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
                                NULL);

    if (hFile == INVALID_HANDLE_VALUE) {
        LPSTR message = "Error occurred! File is not existed";
        prnt(message);
        return ERROR_CODE;
    }

    if (N <= len) {
        buf[N+1] = '\0';
    }

    // GetProcessHeap() - Извлекает дескриптор кучи вызывающего
процесса
    // Выделенная память будет инициализирована до нуля
    LPVOID pBuf = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY,
strlen(buf)*sizeof(char));
    if (!pBuf) {
        LPSTR message = "HeapAlloc failed";
        prnt(message);
    }

    pBuf = buf;

    // Записывает данные в указанный файл
    // Указатель на структуру с параметрами асинхронной записи
    if (!WriteFile(hFile, pBuf, strlen(pBuf),
                    &dwBytesWritten, NULL)) {
        LPSTR message = "Error occurred! Data is not written";
        prnt(message);
        return ERROR_CODE;
    }

```

```

// Закрывает дескриптор открытого объекта
CloseHandle(hFile);

// Освобождает блок памяти, выделенный из кучи
HeapFree(GetProcessHeap(), 0, pBuf);
free(buf);
return SUCCESS;
}

```

ПРИЛОЖЕНИЕ Б.

Программа для операционной системы Linux

```

#include "fcntl.h"
#include "stdlib.h"
#include "string.h"
#include "unistd.h"

#define ERROR_CODE 1
#define SUCCESS 0
#define SIZE 128

int main(int argc, char *argv[]){

    if (argc < 3){
        char *message = "It is needed 3 arguments\n";
        write(STDOUT_FILENO, message, strlen(message));
        return ERROR_CODE;
    }

    if (argc > 3){
        char *message = "Expected 3 arguments\nLast argument should be
in \"\"\"\"";
        write(STDOUT_FILENO, message, strlen(message));
        return ERROR_CODE;
    }
}

```



```

char *buf = malloc(SIZE*sizeof(char));
char *s = "Input string to 128 chars\n";
write(STDOUT_FILENO, s, strlen(s));
int len = read(STDIN_FILENO, buf, SIZE)-1;

if (len == 0){
    char *message = "Read error: cannot read from console";
    write(STDOUT_FILENO, message, strlen(message));
    return ERROR_CODE;
}

int N = atoi(argv[1]);

if (N < 0){
    char *message = "N must be a positive number";
    write(STDOUT_FILENO, message, strlen(message));
    return ERROR_CODE;
}

int file = open(argv[2], O_WRONLY);

if (file < 0){
    char *message = "Error occurred! File is not existed\n";
    write(STDOUT_FILENO, message, strlen(message));
    return ERROR_CODE;
}

if (N <= len){
    buf[N] = '\0';
}

if (!write(file, buf, strlen(buf))){
    char *message = "Error occurred! Data is not written\n";
    write(STDOUT_FILENO, message, strlen(message));
    return ERROR_CODE;
}

free(buf);

close(file);

return SUCCESS;
}

```

ПРИЛОЖЕНИЕ В.

Текст задания

Программа должна считать с консоли строку текста и записать первые N её символов в существующий файл. Число N передается в качестве первого аргумента командной строки. Имя файла передается в качестве второго аргумента командной строки. Если число N больше количества имеющихся (введенных) данных, необходимо записать те данные, которые доступны. Если файл не существует, программа должна вывести сообщение об ошибке и не осуществлять операцию записи.