Федеральное государственное бюджетное образовательное учреждение высшего

образования

«Чувашский государственный университет им. И.Н. Ульянова»

Кафедра вычислительной техники

# **Лабораторная работа 8.**

**ФАЙЛОВЫЙ ВВОДВЫВОД В WIN32**

Вариант 9

Выполнил: Иванов В.С.

студент группы ИВТ-41-22

Проверила:

Доцент Андреева А.А.

Чебоксары, 2024

Цель работы:изучить функции Win API , применить полученные знания на практике.

Задание.

Создать заданную папку в текущей. Имя папки задается в командной строке.

Описание используемых API-функций.

Функция CreateDirectory. С помощью данной функции можно создать каталог. Функция имеет два параметра. Первый параметр указывает на строку, содержащую имя создаваемого каталога, второй — на атрибут безопасности (см. подробнее далее). Обычно второй параметр полагают равным нулю.

**Параметры**

[in] lpPathName

Путь к созданному каталогу.

[in, optional] lpSecurityAttributes

Указатель на структуру SECURITY\_ATTRIBUTES. Элемент lpSecurityDescriptor указывает дескриптор безопасности для нового каталога. Если lpSecurityAttributesNULL, каталог получает дескриптор безопасности по умолчанию. Списки управления доступом в дескрипторе безопасности по умолчанию для каталога наследуются от родительского каталога.

**Возвращаемое значение**

Если функция выполнена успешно, возвращаемое значение ненулевое.

Если функция завершается ошибкой, возвращаемое значение равно нулю.

Есть API-функция GetCommandLine, которая возвращает указатель на командную строку. Эта функция одинаково работает как для консольных приложений, так и для приложений GUI.

**Возвращаемое значение** — указатель на строку командной строки для текущего процесса.

Особо отмечу, что при вызове функции API может возникнуть ошибка, связанная либо с неправильными входными параметрами, либо невозможностью получения искомых результатов из-за определенного состояния системы. В этом случае индикатором будет содержимое, возвращаемое в регистре EAX. К сожалению, для разных функций это может быть разное значение. Это может быть 0 (чаще всего), –1 или какое-либо другое ненулевое значение. Напомню в этой связи, что 0 во многих языках программирования считается синонимом значения FALSE, а значению TRUE ставится в соответствие 1. В каждом конкретном случае следует свериться с документацией. С помощью функции GetLastError можно получить код последней происшедшей ошибки, т. е. расшифровку того, почему функция API выполнилась не должным образом.

**Возвращаемое значение** — это код последней ошибки вызывающего потока.

Каждая программа в Win32 в конце своего выполнения ОБЯЗАТЕЛЬНО должна вызвать функцию ExitProcess. После выполнения последней инструкции программы, дальше идёт пустота, даже если там что-то осмысленное, все равно у выделенной памяти есть конец, а после конца идёт невыделенная (т.е. несуществующая) память. Процессор начнёт выполнять непонятно что, пойдут глюки.

**Параметры**

[in] uExitCode

Код выхода для процесса и всех потоков.

**Возвращаемое значение** Нет

GetStdHandle получает дескриптор буфера вывода консоли.

**Параметры**

*nStdHandle* [в]  
стандартном устройстве. Этот параметр может принимать одно из следующих значений.

| **Значение** | **Значение** |
| --- | --- |
| **STD\_INPUT\_HANDLE STD\_INPUT\_HANDLE** ((DWORD)-10) | Стандартное устройство ввода. Изначально это буфер ввода консоли, CONIN$. |
| **STD\_OUTPUT\_HANDLE - ДЕСКРИПТОР** ((DWORD)-11) | Стандартное устройство вывода. Изначально это активный буфер экрана консоли, CONOUT$. |
| **STD\_ERROR\_HANDLE - ОБРАБОТЧИК STD\_ERROR\_HANDLE** ((DWORD)-12) | Стандартное устройство вывода ошибок. Изначально это активный буфер экрана консоли, CONOUT$. |

**Возвращаемое значение**

Если функция выполнена успешно, возвращаемое значение представляет собой дескриптор указанного устройства или перенаправленный дескриптор, заданный предыдущим вызовом **[SetStdHandle](https://learn.microsoft.com/en-us/windows/console/setstdhandle)**. Дескриптор имеет права доступа **GENERIC\_READ** и **GENERIC\_WRITE**, если только приложение не использовало **SetStdHandle** для установки стандартного дескриптора с меньшими правами доступа

Функция WriteFile выполняет запись текста сообщения (например, о результате выполнения программы) непосредственно в стандартный вывод (консоль). Это позволяет пользователю увидеть соответствующее сообщение:

* Успешное создание папки: Folder created successfully!
* Ошибка при создании папки: Error creating folder.
* Завершение программы: Exiting program.

**Параметры**

[in] hFile

Дескриптор устройства ввода-вывода или файла (например, файл, поток файлов, физический диск, том, буфер консоли, ленточный диск, сокет, ресурс связи, mailslot или канал).

[in] lpBuffer

Указатель на буфер, содержащий данные, записываемые в файл или устройство.

[in] nNumberOfBytesToWrite

Количество байтов, записываемых в файл или устройство.

[out, optional] lpNumberOfBytesWritten

Указатель на переменную, получающую количество байтов, записанных при использовании синхронного параметра hFile. WriteFile задает значение нулю перед выполнением любой проверки ошибок или работы.

[in, out, optional] lpOverlapped

Указатель на структуру OVERLAPPED требуется, если был открыт параметр hFile с FILE\_FLAG\_OVERLAPPED, в противном случае этот параметр может быть NULL.

**Возвращаемое значение**

Если функция выполнена успешно, возвращаемое значение ненулевое (**TRUE**).

Если функция завершается сбоем или выполняется асинхронно, возвращаемое значение равно нулю (**FALSE**). Чтобы получить расширенные сведения об ошибке, вызовите функцию getLastError .

Код программы:

include typefile.inc

.386

.model FLAT, STDCALL

.data

nameout db 'CONOUT$', 0   ; Имя устройства для вывода (консоль)

folderName db 100 dup(0)  ; Буфер для имени создаваемой папки

msgSuccess db 'Folder created successfully!', 0Dh, 0Ah, 0

msgError db 'Error creating folder.', 0Dh, 0Ah, 0

msgEnd db 'Exiting program.', 0Dh, 0Ah, 0

bytesWritten dd 0         ; Количество записанных байт

.code

\_start:

    call GetCommandLine   ; Получаем указатель на командную строку

    mov esi, eax          ; Сохраняем указатель на строку команд в регистре ESI

    xor ecx, ecx          ; Обнуляем счетчик параметров (ECX)

    mov edx, 1            ; Устанавливаем признак наличия параметра (EDX)

n1:

    cmp byte ptr [esi], 0 ; Проверяем конец строки

    je end\_               ; Если конец строки, переходим к завершению

    cmp byte ptr [esi], 32 ; Сравниваем текущий символ с пробелом

    je n3                 ; Если пробел, переходим к обработке пробела

    add ecx, edx          ; Увеличиваем счетчик параметров (ECX)

    cmp ecx, 2            ; Проверяем, является ли это вторым параметром

    je n4                 ; Если да, переходим к копированию имени папки

    xor edx, edx          ; Обнуляем признак наличия параметра

    jmp n2                ; Переходим к следующему символу

n3:

    or edx, 1             ; Устанавливаем признак наличия параметра

n2:

    inc esi               ; Переходим к следующему символу в строке

    jmp n1                ; Возвращаемся к началу цикла

n4:

    ; Копируем имя папки в буфер folderName

    mov edi, offset folderName ; Устанавливаем указатель на буфер для имени папки

    mov ecx, 100           ; Устанавливаем максимальную длину копируемой строки

copy\_loop:

    cmp byte ptr [esi], 0  ; Проверяем конец строки

    je create\_directory    ; Если конец строки, переходим к созданию директории

    mov al, [esi]          ; Копируем текущий символ в AL

    stosb                  ; Сохраняем символ в буфер folderName

    inc esi                ; Переходим к следующему символу в строке

    jmp copy\_loop          ; Возвращаемся к началу цикла копирования

create\_directory:

    push NULL

    push offset folderName

    call CreateDirectory   ; Вызов CreateDirectory

    cmp eax, 0             ; Проверяем результат выполнения функции

    jne write\_success      ; Если результат не равен нулю, значит создание успешно

    call write\_message, offset msgError   ; Выводим сообщение об ошибке

    jmp write\_end

write\_success:

    call write\_message, offset msgSuccess ; Выводим сообщение об успешном создании

write\_end:

    call write\_message, offset msgEnd     ; Выводим сообщение о завершении

end\_:

    push 0

    call ExitProcess       ; Завершаем программу

; Процедура для вывода сообщения

write\_message proc

    push ebp

    mov ebp, esp

    pushad                 ; Сохраняем регистры

    ; Получаем дескриптор консоли

    push STD\_OUTPUT\_HANDLE

    call GetStdHandle

    mov ebx, eax           ; Сохраняем дескриптор

    ; Вычисляем длину строки

    mov eax, [ebp+8]       ; Адрес сообщения

    mov edx, eax

    xor ecx, ecx

find\_length:

    cmp byte ptr [edx], 0  ; Конец строки?

    je write\_string        ; Если 0, строка завершена

    inc edx

    inc ecx

    jmp find\_length

write\_string:

    push NULL              ; lpOverlapped = NULL

    push offset bytesWritten ; Адрес переменной для записанных байт

    push ecx               ; Длина строки

    push eax               ; Указатель на сообщение

    push ebx               ; Дескриптор консоли

    call WriteFile         ; Вызов WriteFile

    popad                  ; Восстанавливаем регистры

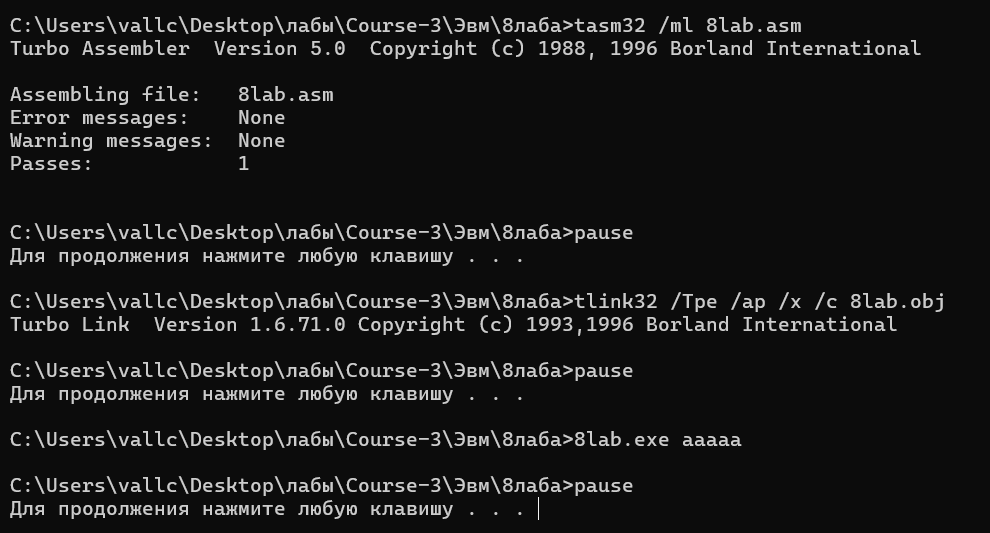
    mov esp, ebp

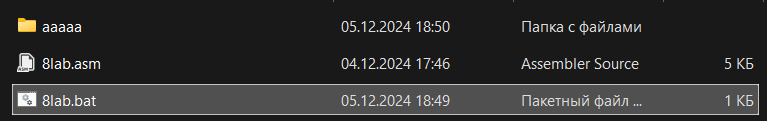
    pop ebp

    ret

write\_message endp

end \_start

При вводе в .bat 8lab.exe aaaaa в корневой папке создастся папка с таким названием  




Вывод: изучил функции Win API , применил полученные знания на практике.