Федеральное государственное бюджетное образовательное учреждение высшего

образования

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

Кафедра вычислительной техники

# **Лабораторная работа 4.**

**СОЗДАНИЕ ДИНАМИЧЕСКИХ БИБЛИОТЕК**

Вариант 9

Выполнил: Иванов В.С.

студент группы ИВТ-41-22

Проверила:

Доцент Андреева А.А.

Чебоксары, 2024

Цель работы:

Изучить основные сведения об использовании DDL библиотек при их практическом использовании на языке ассемблера

Задание:

Разработать динамическую библиотеку, реализующую функции в соответствии с заданным вариантом, и программу для демонстрации ее возможностей. Использовать как явное, так и неявное связывание.

9. function PadCh(S: pchar; C: char; Len: byte): pchar. Возвращает строку, в которой S смещена влево, а остаток строки заполнен символами С. Для этого знаки С включаются справа от конца S до тех пор, пока общая длина строки не станет равной Len. Если S длиннее чем Len, то строка не изменяется. Если S - пустая строка, то возвращается строка из Len символов С.



Текст подпрограммы:  
.386

.model flat

public PadCh

.code

start@12:

mov al,1

ret 12

;function PadCh(S: string; C: char; Len: byte): string,

;Возвращает строку, в которой S смещена влево, а остаток строки заполнен символами С.

;Для этого знаки С включаются справа от конца S до тех пор пока общая длина строки не

;станет равной Len. Если S длиннее чем Len, то строка не изменяется.

;Если S пустая строка, то возвращается строка из Len символов С.

PadCh proc near

; адреса параметров в стеке:

S equ dword ptr [ebp+16] ; адрес строки S:string

Ch1 equ byte ptr [ebp+12] ; адрес параметра С :Char

Len equ byte ptr [ebp+8] ; адрес параметра Len:Byte

Res equ dword ptr [ebp+20] ; адрес строки результата

push ebp ; сохранение bp

mov ebp, esp ; настройка bp на вершину стека

;push ds ; сохранение ds

mov edi, [Res] ; es:di:=адрес результата

mov esi, [S] ; ds:si:=адрес исходной

; строки

mov ecx,0

cld ; очистка флага направления (инкремент)

lodsb ; al:=(ds:[si]), si:=si+1 (al - длина S)

stosb ; копируем длину строки S в Res

mov ah, al ; сохраняем длину строки S

cmp al, [Len] ; сравниваем длину S с Len

jae StoreLen ; если S >= Len, то копируем S и выходим

Pad:

mov al, [Len]

mov [edi-1], al ; Дополняем строку

mov cl, ah

rep movsb ; записать очередной символ результата Res

StoreLen:

mov cl, [Len]

sub cl, ah ; S длиннее, чем Len

mov al, [Ch1] ; добавляем очередной символ

;Копируем строку

rep stosb

Exit: ; pop ds ; восстановить ds

pop ebp ; восстановить bp

ret 16 ; выход с удалением параметров Ch1,

; Ch2 и адреса S (Res удалять нельзя!)

PadCh endp

end start@12  
  
Текст основной программы с неявным преобразованием:  
include include.inc

includelib 4\_kab\_d.lib

extrn PadCh: near ; Объявление внешней подпрограммы

.386

.model flat, stdcall

.const

title\_S\_before db 'Before neyav add:', 0

title\_S\_after db 'After add:', 0

.data

S db 5, 'qwert', 0 ; Исходная строка с длиной

Res db 255 dup(0) ; Буфер для результата, инициализированный нулями

Ch1 db 'A' ; Символ для добавления

Len1 db 5 ; Длина строки (должна соответствовать фактической длине)

.code

start:

; Вывод сообщения перед добавлением

call MessageBox, 0, offset S, offset title\_S\_before, MB\_OK

; Вызов вашей подпрограммы PadCh

push offset Res ; адрес результата

push offset S ; адрес строки S

mov al, [Ch1] ; символ для добавления

push eax ; символ

mov al, [Len1] ; длина строки

push eax ; длина

call PadCh ; вызов вашей подпрограммы

; Вывод сообщения после добавления

call MessageBox, 0, offset Res, offset title\_S\_after, MB\_OK

; Завершение программы

push 0 ; код завершения

call ExitProcess ; завершение программы

ends

end start  
  
Текст основной программы с явным преобразованием:  
include include.inc

.386

.model flat, stdcall

.const

title\_S\_before db 'Before yav add:', 0

title\_S\_after db 'After add:', 0

libr db '4\_kab\_d.dll',0

nameproc db 'PadCh',0

.data

S db 5, 'qwert', 0 ; Исходная строка с длиной

Res db 255 dup(0) ; Буфер для результата, инициализированный нулями

Ch1 db 'A' ; Символ для добавления

Len1 db 5 ; Длина строки (должна соответствовать фактической длине)

hlib dd ?

PadCh dd ?

.code

start:

call LoadLibrary,offset libr ;загрузка библиотеки

mov hlib,eax

call GetProcAddress,hlib,offset nameproc ;получение адреса функции

mov PadCh,eax

; Вывод сообщения перед добавлением

call MessageBox, 0, offset S, offset title\_S\_before, MB\_OK

; Вызов вашей подпрограммы PadCh

push offset Res ; адрес результата

push offset S ; адрес строки S

mov al, [Ch1] ; символ для добавления

push eax ; символ

mov al, [Len1] ; длина строки

push eax ; длина

call PadCh ; вызов вашей подпрограммы

; Вывод сообщения после добавления

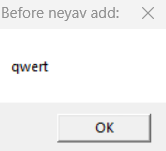
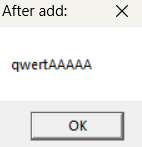
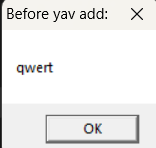
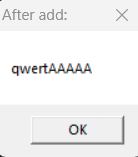
call MessageBox, 0, offset Res, offset title\_S\_after, MB\_OK

; Завершение программы

push 0 ; код завершения

call ExitProcess ; завершение программы

ends

end start  
  
Результаты кода:  
     
  
Вывод: изучил основные сведения об использовании DDL библиотек при их практическом использовании на языке ассемблера

Ответы на вопросы:  
**1.Все командные строки**

При неявном связывании:

include include.inc

includelib 4\_kab\_d.lib

extrn PadCh: near

call MessageBox, 0, offset S+1, offset title\_S\_before, MB\_OK

call PadCh

call MessageBox, 0, offset Res+1, offset title\_S\_after, MB\_OK

call ExitProcess

При явном связывании:

include include.inc

libr db '4\_kab\_d.dll',0

nameproc db 'PadCh',0

call LoadLibrary,offset libr

mov hlib,eax

call GetProcAddress,hlib,offset nameproc

mov PadCh,eax

call MessageBox, 0, offset S+1, offset title\_S\_before, MB\_OK

call PadCh

call MessageBox, 0, offset Res+1, offset title\_S\_after, MB\_OK

call ExitProcess

**2. Содержание секции экспорта dll файла.**

MZP яя ё @ є

ґ Н!ёLН!ђђThis program must be run under Win32

$7 PE L 4Y±K а ЋЎ

@

0 D CODE `.edata @ @ °ВU‹м‹}‹u№ ь¬ЄЉа:Er

ђђђђЉИу¤лђђђЉE€GяЉМу¤ЉM\*МЉEуЄ]В 2 ( , 0 > 4\_kab\_d.DLL PadCh

MZ – подпись DOS-заголовка, указание что файл является dll файлом

This program must be run under Win32 – сообщение при запуске фала в среде DOS

PE L4Y,K  
PE – подпись заголовка PE, указывает на начало структуры Pe-файла

После PE идут поля заголовка, размер кода, колво секций, адрес точки входа, флаги

Таблицы импортов и экспортов  
CODE `.edata @ @

4\_kab\_d.DLL PadCh

.edata: Секция экспортов. Здесь хранятся функции, которые экспортируются данной DLL.

4\_kab\_d.DLL PadCh : Название экспортируемой функции или указатель на неё

**3. Содержимое lib файла**

р Ђ PadChґ€  PadCh  
4\_kab\_d.DLL ^€ ы tЉ t с­ a PadCh

PadChґ€  - содержит служебную информацию о библиотеке

PadCh 4\_kab\_d.DLL – имя экспортируемой функции и указатель на библиотеку из которой происходит экспорт

^€ ы tЉ t с­ - метаданные, подсказки для компоновщика

**4.Содержимое секции импорта exe файла**

MZP яя ё @ є

ґ Н!ёLН!ђђThis program must be run under Win32

$7 PE L 4YІK а ЋЃ

@

P 0 @ ( CODE `DATA @ А.idata 0

@ А.reloc @ @ P j h!@ h @ j и5 h @ h @  !@ P  !@ Pи8 j h!@ h @ j и j и я%t0@ я%|0@ я%Ђ0@ я%„0@ я%€0@ я%ђ0@ qwert A

Before neyav add: After add: P0 0 t0 X0 Ј0 |0 l0 °0 ђ0 ј0 К0 Ш0 и0 ц0 1 ј0 К0 Ш0 и0 ц0 1 USER32.dll KERNEL32.dll 4\_kab\_d.DLL MessageBoxA ExitProcess LoadLibraryA FreeLibrary GetProcAddress PadCh ( 0000‑0$01060J0P0V0\0b0h0

Импортируемые библиотеки:

USER32.dll: Библиотека, содержащая функции для работы с пользовательским интерфейсом (например, создание окон, диалогов и т. д.).

KERNEL32.dll: Библиотека, содержащая основные функции операционной системы (например, работа с памятью, процессами и т. д.).

4\_kab\_d.DLL

Импортируемые функции:

MessageBoxA: Функция для отображения сообщения в окне.

ExitProcess: Завершает выполнение процесса.

LoadLibraryA: Загружает библиотеку в адресное пространство процесса.

FreeLibrary: Освобождает загруженную библиотеку.

GetProcAddress: Получает адрес функции в загруженной библиотеке.

PadCh

**5. Как формируется секция импорта?**

Во время линковки линковщик разрешает внешние ссылки на функции и формирует секцию импорта. Линковщик использует .lib файл, чтобы определить, какие функции исполняемый файл будет импортировать и из каких DLL.

**6. Что происходит при загрузке exe файла при выполнении, если в нем содержится секция импорта?**

Когда операционная система загружает исполняемый файл (EXE) с секцией импорта:

1. Загрузка EXE-файла в память

* Чтение заголовка: Операционная система считывает заголовок PE, который содержит информацию о структуре файла, включая адреса секций, точки входа и размер секций.
* Создание процесса: ОС создает новый процесс в памяти и выделяет необходимое количество памяти.

2. Обработка секции импорта

* Инициализация секции импорта: Операционная система ищет в заголовке PE секцию импорта, которая содержит список всех импортируемых библиотек и функций.

2.1. Поиск библиотек

* Поиск DLL: ОС проходит по списку импортируемых библиотек. Например, если секция импорта содержит USER32.dll, ОС ищет эту библиотеку в системных каталогах

2.2. Загрузка библиотек

* Загрузка в память: При нахождении библиотеки ОС загружает её в память, если она еще не была загружена. Если библиотека уже загружена, ОС просто использует её существующий экземпляр.

3. Разрешение адресов функций

* Инициализация таблицы импортов: Для каждой импортируемой функции (например, MessageBoxA) ОС получает её адрес в загруженной библиотеке.
* Заполнение таблицы: ОС заполняет таблицу импортов в исполняемом файле реальными адресами функций. Это позволяет программе вызывать функции как будто они находятся в её собственном коде.

4. Выполнение точки входа

* Переход к точке входа: После завершения всех операций с импортами ОС передает управление к точке входа программы

5. Выполнение программы

* Вызовы функций: Программа может теперь вызывать импортированные функции. Когда программа вызывает, например, MessageBoxA, она использует адрес, полученный на предыдущем этапе.

6. Завершение работы

* Завершение работы: Когда программа завершает выполнение (например, вызывает ExitProcess), ОС освобождает ресурсы, связанные с процессом, включая память, используемую для импортированных библиотек.

**7.Почему считается, что явное связывание более гибкое?**

Явное связывание (explicit linking) — это процесс, при котором программа сама динамически загружает DLL и получает адреса функций во время выполнения с помощью API, таких как LoadLibrary и GetProcAddress.  
  
Неявное связывание (implicit linking) — это процесс, при котором все необходимые DLL загружаются автоматически при запуске программы, и таблица импорта заполняется загрузчиком.  
  
Явное связывание более гибкое, потому что:  
  
Программа может контролировать, когда и какие DLL загружать.  
Возможно загружать разные версии DLL в зависимости от условий (например, проверять совместимость перед загрузкой).  
Позволяет обрабатывать ошибки при загрузке DLL или отсутствии функций, в то время как неявное связывание приводит к немедленному завершению работы программы в случае ошибки.  
Можно загружать DLL по запросу, что позволяет экономить ресурсы программы.  
Таким образом, явное связывание даёт разработчику больше контроля над процессом загрузки библиотек и управлением их использованием.