**Ассемблер под Windows для чайников. Часть 2**

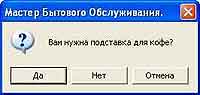
[главная](file:///C:\kg\) » [статьи](file:///C:\kg\arch.html) » [2008](file:///C:\kg\2008\) » [18](file:///C:\kg\2008\18\) » 08.05.2008‣программирование

В прошлый раз мы познакомились с компилятором FASM для Windows — рассмотрели основы его синтаксиса и написали нашу первую программу. Самые любопытные уже, наверное, заглянули в папку EXAMPLES и обнаружили там с дюжину готовых примеров различного уровня сложности. Если вы еще не сделали этого — быстренько открывайте ..\FASM\EXAMPLES\ и изучайте — даю вам 5 минут на это! Время пошло.  
  
 Ну что же, теперь мы можем приступать к очередной тренировке ввода с клавиатуры букв, цифр и знаков препинания. Ленивые могут скопировать исходный код из интернета: [сайт](http://forum.sa-sec.org/index.php?showtopic=766) Помните анекдот про подставку для кофе? Когда секретарша звонит сисадмину и сообщает, что на ее компьютере сломалась подставка для кофе. Админ, не раздумывая, набирает другой номер и говорит: "Петрович, у секрятаря CD-ROM накрылся, надо заменить". Вот мы сейчас и оформим программку управления подставкой для кофе. За основу взят пример, идущий в комплекте с виндовской версией

фасма.

**format PE GUI 4.0  
  
include 'win32a.inc'  
  
; секции не обозначены, поэтому fasm автоматически создаст секцию .flat  
; в которой разместятся и код, и данные, что позволит уменьшить размер файла  
  
invoke MessageBoxA,0,\_message,\_caption,MB\_ICONQUESTION+MB\_YESNOCANCEL  
cmp eax,IDNO  
je close  
cmp eax,IDYES  
jne exit  
  
;open:  
invoke mciSendString,\_cd\_open,0,0,0  
jmp exit  
  
close:  
invoke mciSendString,\_cd\_close,0,0,0  
  
exit:  
invoke ExitProcess,0  
  
\_message db 'Вам нужна подставка для кофе?',0  
\_caption db 'Мастер Бытового Обслуживания.',0  
  
\_cd\_open db 'set cdaudio door open',0  
\_cd\_close db 'set cdaudio door closed',0  
  
; импортируемые данные разместятся в этой же секции:  
  
data import  
  
library kernel32,'KERNEL32.DLL',\  
user32,'USER32.DLL',\  
winmm,'WINMM.DLL'  
  
import kernel32,\  
ExitProcess,'ExitProcess'  
  
import user32,\  
MessageBoxA,'MessageBoxA'  
  
import winmm,\  
mciSendString,'mciSendStringA'  
  
end data**

Вот такое окошко должно у нас получиться.



Нажмете кнопку "Да" — лоток CD-ROM'а откроется. Нажмете "Нет" — закроется. А сейчас — традиционное разбирательство по вопросу "Как Это Работает":  
 Символ "точка с запятой" (;) означает, что в строке или в оставшейся ее части размещен комментарий. Когда компилятор встречает этот символ, то игнорирует текст, идущий после точки с запятой, и переходит к обработке следующей строки. Исключением из этого правила является точка с запятой, заключенная в кавычки. Старайтесь всегда вносить в код пояснения — в будущем это поможет вам не запутаться в собственных программах и упростит понимание ваших текстов другими людьми. Как вы могли понять из комментария, в этом примере секция кода и данных будет скомпилирована в одну секцию. В нашем случае это положительно скажется на размере исполняемого файла, точнее, отрицательно — короче, размер файла будет меньше. Каждая отдельная секция файла для ускорения доступа к ней операционной системы округляется до 512 байт. Даже если секция будет содержать лишь пару байт данных или кода, компилятор все равно допишет недостающее число нулевых байт. Значит, если у нас совсем немного кода и данных, мы можем разместить их в одной универсальной секции для уменьшения размера получаемого файла. Зачем вообще нужны эти секции? Ну, типа для повышения надежности и безопасности программного обеспечения. Каждая секция при запуске программы получает свой набор прав. Обычно секция данных может быть прочитана и записана, но не может быть исполнена, а секция кода имеет разрешение на исполнение, но не может быть перезаписана. Но это все формальности. При желании можно найти и способы изменения исполняемого кода, и исполнения команд прямо из секции данных. Однако не будем забегать слишком далеко вперед и продолжим разбор нового материала.  
Что такое invoke MessageBox, вы знаете из предыдущего занятия. Однако здесь вы видите MessageBoxA. Не бойтесь, это ведь та же самая API-функция. Просто в тот раз мы импортировали ее командой import USER32,MessageBox,'MessageBoxA', а теперь — командой import user32,MessageBoxA,'MessageBoxA'. Название импортируемой функции, заключенное в кавычки, должно быть точным: оно будет передано операционной системе в момент запуска программы для получения адреса функции. А вот псевдоним, стоящий через запятую перед именем функции, может отличаться от имени — например: import USER32, Box, 'MessageBoxA'. Только в таком случае вы усложните понимание кода себе и другим людям. Так что не принимайте это за сигнал к действию, а просто имейте в виду, что псевдоним и реальное имя функции могут иногда различаться. Зачем же я изменил псевдоним этой функции и заострил на этом ваше внимание? Дело в том, что, читая код программ, написанных другими людьми, вы можете встретить как первый, так и второй вариант. А в редких случаях и третий, и еще какой-нибудь четвертый. Вообще функции Windows, работающие с текстовыми строками, бывают двух типов: A (кодировка ANSI) и W (кодировка Юникод). Мы в основном будем работать с кодировкой ANSI, но вам следует знать, что у каждой API-функции, использующей текст ANSI, есть брат-близнец для кодировки Unicode. Итак, вызов функции MessageBox выводит окно с сообщением и приостанавливает работу программы, ожидая реакцию пользователя. По завершении функция возвращает программе код нажатой пользователем кнопки или возвращает 0, если не хватило памяти для создания окна с сообщением. Возвращаемые значения могут быть следующими:

|  |  |  |
| --- | --- | --- |
| Псевдоним | Значение | Нажатая кнопка |
| IDOK | 1 | OK |
| IDCANCEL | 2 | Отмена (Cancel) |
| IDABORT | 3 | Прервать (Abort) |
| IDRETRY | 4 | Повтор (Retry) |
| IDIGNORE | 5 | Пропустить (Ignore) |
| IDYES | 6 | Да (Yes) |
| IDNO | 7 | Нет (No) |

Так как же нам узнать, какая кнопка была нажата? Где найти это "возвращаемое значение", и под каким соусом его подавать на стол? Будем разбираться. В процессоре существуют ячейки высокоскоростной памяти, которые физически находятся вблизи его ядра. Эти ячейки называются регистрами. Если вы уже сейчас захотите узнать об этих регистрах более подробно, воспользуйтесь поиском в интернет (ключевые слова: регистры процессора). Однако на данный момент вам может хватить и приведенной здесь информации о регистрах. Основных регистров пользователя всего четыре: EAX(Accumulator), EBX(Base), ECX(Count), EDX(Data). Каждый из них имеет размер 4 байта (32 бита) и может использоваться в вычислениях целиком или частично. Например, можно обратиться к целому регистру EAX, можно работать с его младшей половинкой AX и даже с четвертинками AH и AL. Нельзя напрямую отдельно обратиться к старшей половинке регистра EAX, поэтому у нее нет собственного имени.

|  |  |  |
| --- | --- | --- |
| EAX (4 байта) |  |  |
|  | AX (2 байта) |  |
|  | AH(1) | AL(1) |

Аналогично устроены и регистры EBX, ECX, EDX, а их части называются соответственно BX/BH/BL, CX/CH/CL, DX/DH/DL. Существуют и другие регистры процессора, но мы будем говорить о них по ходу их появления в наших примерах. Большинство функций Windows возвращают результаты своих действий в регистр процессора EAX. Функция MessageBox поступает так же: после того, как пользователь нажмет на кнопку в окне с сообщением, она поместит в регистр EAX числовое значение нажатой кнопки. Теперь нам надо в зависимости от полученного значения выполнить то или иное действие. Для этого мы будем использовать команду сравнения (CMP) и команды условного перехода (JE и JNE).  
CMP — сокращение от Compare (Сравнить). Синтаксис этой команды: [CMP приемник, источник]. Она сравнивает два числа, вычитая источник из приемника, не изменяя их содержимое.  
JE — Jump if Equal (Переход, если равно). JNE — Jump if Not Equal (Переход, если не равно). Это команды-антонимы — они противоположны по значению. Синтаксис JE, JNE и других команд перехода такой: [JE метка]. Команды условного перехода используются после команд CMP и SUB (вычитание с сохранением результата в приемник). Переход на метку осуществляется только если соблюдено условие перехода. Если условие не соблюдено, то программа продолжает выполняться в обычном порядке. Команда JMP (Jump) является командой безусловного перехода, то есть переход на указанную метку осуществляется в любом случае. В нашей программе мы сначала сравниваем содержимое eax и IDNO (эквивалент числа 7) и переходим на метку close, если они равны (JE). Иначе — сравниваем eax и IDYES (эквивалент числа 6) и переходим на метку exit, если они не равны (JNE). По этой логике программа выполнит строки  
  
**invoke mciSendString,\_cd\_open,0,0,0  
jmp exit**

только при условии, что содержимое eax после функции MessageBox будет равно IDYES. Если eax=IDNO, то выполняются команды после метки close включая команды после метки exit. Если результат не равен ни IDYES, ни IDNO, то выполняется вызов функции завершения работы программы: invoke ExitProcess,0. API-функция mciSendString отправляет командную строку устройству MCI (Media Control Interface). Устройство, которому отправляется команда, должно быть определено в командной строке. У функции четыре параметра: указатель на командную строку, указатель на буфер для ответа, размер буфера для ответа (количество символов), хэндл окна, которому отправляется напоминание при ответе (для этого в командной строке должен присутствовать параметр notify). В нашем случае мы обойдемся без получения ответов от устройства, поэтому вместо последних трех параметров ставим нолики. А вот первый параметр — это целый раздел в интернет-библиотеке мелкомягких (MSDN), поэтому я приведу вам лишь общую информацию о командных строках мультимедиа. Если вы знаете английский, то можете ознакомиться с полной версией описания этих команд по адресу: [сайт](http://msdn2.microsoft.com/en-us/library/ms712587.aspx) Командная строка MCI состоит из трех основных частей: команда, устройство, параметры. Эти части должны разделяться пробелами. Команда в нашем конкретном случае — это set. Бывают команды open, play, close и другие. Устройство у нас — cdaudio. Устройством также может являться полное имя файла, псевдоним, установленный параметром alias предшествующей команды open, слово new в команде open при открытии устройства на запись, слово all — для отправки команды всем открытым в программе устройствам. Параметры разделяются пробелами, перечисляются в произвольном порядке и могут вообще отсутствовать в некоторых командах. Теперь мы можем добавить к нашей сегодняшней программе звуковое сопровождение. Для этого необходимо дописать строку  
**invoke mciSendString,\_wav\_play,0,0,0**  
прямо перед вызовом функции MessageBox и строку  
**\_wav\_play db 'play c:\windows\media\tada.wav',0**  
где-нибудь после вызова функции ExitProcess, но перед импортом данных. Если у вас папка с windows находится в другом месте — укажите свой путь. Можете вообще указать путь к другому wav-файлу или даже попробовать другие форматы: все зависит от кодеков, установленных в вашей windows. На моей системе прокатило даже воспроизведение видео! Подобным образом вы можете озвучить открытие и закрытие лотка дискового привода. Только имейте в виду, что, если программа завершит работу раньше, чем закончится выбранный вами звук, то воспроизведение прервется тоже. Например, если поставить строку с командой воспроизведения прямо перед выходом из программы (ну перед invoke ExitProcess), то вы вообще не услышите никакого звука. Он попросту не успеет начаться, когда ему уже пора будет заканчиваться. Для такого случая предусмотрен параметр wait. Если в командной строке мультимедиа указан этот параметр, то MCI вернет управление программе только после полного исполнения команды. Будьте осторожны: если звук будет слишком длинный, вы рискуете надолго "подвесить" вашу программу в ожидании окончания воспроизведения. Пример строки с использованием параметра wait:  
**\_wav\_play db 'play c:\windows\media\tada.wav wait',0**  
Ну, а теперь переделаем нашу программку в полезную утилиту. Программа будет проверять статус лотка CD-ROM'а (открыт/закрыт) и изменять его на противоположный. В таком случае мы сможем вывести на рабочий стол ярлык для нашей программы, выбрать для него какой-нибудь подходящий значок (в свойствах ярлыка) и использовать его почти как кнопку "открыть/закрыть" на самом приводе:  
 **format PE GUI 4.0  
  
include 'win32a.inc'  
  
invoke mciSendString,\_cd\_state,\_ret,5,0  
invoke lstrcmp,\_ret,\_ret\_open  
cmp eax,0  
je close  
;open:  
invoke mciSendString,\_cd\_open,0,0,0  
jmp exit  
close:  
invoke mciSendString,\_cd\_close,0,0,0  
exit:  
invoke ExitProcess,0  
  
\_cd\_state db 'status cdaudio mode',0  
\_cd\_open db 'set cdaudio door open',0  
\_cd\_close db 'set cdaudio door closed',0  
\_ret\_open db 'open',0  
\_ret db 5 dup (?)  
  
data import  
  
library kernel32,'KERNEL32.DLL',\  
user32,'USER32.DLL',\  
winmm,'WINMM.DLL'  
  
import kernel32,\  
ExitProcess,'ExitProcess',\  
GetWindowsDirectory,'GetWindowsDirectory',\  
lstrcmp,'lstrcmpA'  
  
import user32,\  
MessageBoxA,'MessageBoxA'  
  
import winmm,\  
mciSendString,'mciSendStringA',\  
PlaySound,'PlaySoundA'  
end data**  
В самом начале мы отправляем запрос о состоянии устройства cdaudio, поэтому указываем буфер для ответа и его размер. API-функция lstrcmp используется для посимвольного сравнения двух текстовых строк. Если строки одинаковые, она возвращает значение ноль. \_ret\_open — это указатель на строку-образец из пяти символов (нуль-терминатор на конце строки тоже считается). \_ret — это указатель на пустой буфер из 5 байт. Только к моменту сравнения он уже не будет пустым: после вызова mciSendString в буфер будет помещено 5 символов из ответа о текущем состоянии CD-ROM'а — точнее, 4 буквы и завершающий строку нолик. Для слова open нам вполне хватит четырех букв, если же ответ будет другой и займет больше символов, нас это не волнует: не open — значит, открываем. Поэтому, если строки равны, и в eax находится ноль, мы переходим на метку close. Иначе — открываем лоток и проходим к выходу.

db 5 dup (?) резервирует место под 5 неизвестных байт. Можно записать это иначе: rb 5 (reserve 5 bytes). Зарезервированные (неинициализированные) данные не занимают место в файле. Они займут свои места только после запуска в оперативной памяти. Их обычно используют, когда значение заранее не известно. Запись вида db 5 dup (1,2) приведет к созданию пяти копий (duplicate) указанной в скобках последовательности байт. Теперь можно щелкнуть правой кнопкой на получившемся экзешнике и выбрать "Отправить — Рабочий стол (создать ярлык)", в свойствах ярлыка выбрать иконку (сменить значок) и назначить комбинацию клавиш для быстрого вызова — например, Ctrl+Shift+C. При таком раскладе можно будет открыть или закрыть CD-ROM, кликнув на ярлык, или одновременным нажатием выбранной комбинации клавиш. Правда, иногда, MCI долго "думает" прежде чем передать команду CD-ROM'у, но тут уж ничего не попишешь с нашими сегодняшними знаниями. Возможно, когда-нибудь вы напишете свой драйвер, который будет работать напрямую с любым устройством, а может, и целую операционную систему. Но сегодня мы прощаемся. Желаю вам успешно разобраться в пройденном материале, и до новых встреч!  
  
Все приводимые примеры были протестированы на правильность работы под Windows XP и, скорее всего, будут работать под другими версиями Windows, однако я не даю никаких гарантий их правильной работы на вашем компьютере.  
  
**BarMentaLisk, q@sa-sec.org SASecurity gr.**

*Компьютерная газета. Статья была опубликована в номере 18 за 2008 год в рубрике*[*программирование*](file:///C:\kg\abc\%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5\)