1 Лабораторная работа №5. Использование Windows API в ассемблерных программах

Цель работы

Научиться писать программы для защищённого режима процессора x86 и вызывать функции Windows API из ассемблерного кода.

Теоретические сведения[1]

Наиболее естественным средством для разработки Windows-программ на языке ассемблера является пакет MASM. Поскольку оба продукта разрабатываются одной компанией, нет необходимости задумываться о вопросах совместимости, а хорошая освещённость работы с MASM в различных руководствах делает этот пакет весьма простым для освоения. Бесплатная версия MASM (интегрирующаяся с бесплатной Visual C++ 2005 Express Edition) доступна с сайта Microsoft.

Особенности API ОС Windows

Вызов системных функций API win32 из программы на ассемблере подчиняется набору соглашений stdcall, ведущему свою родословную в части именования функций - от языка C, а в части передачи аргументов - от языка Pascal. С точки зрения прикладного программиста и с учетом специфики Windows и MASM эти соглашения заключаются в следующем:

- регистр символов в имени функции не имеет значения. Например, функции с именами Foo и foo это одна и та же функция. Здесь мы наблюдаем отличие от реализации MS Visual C++, где компилятор строго отслеживает совпадение регистра в именах используемых программистом функций с прототипами, содержащимися в системных заголовочных файлах. Компоновщику же, как видим, регистр символов безразличен;

- аргументы передаются вызываемой функции через стек. Если аргумент укладывается в 32-битное значение и не подлежит модификации вызываемой функцией, он обычно записывается в стек непосредственно. В остальных случаях программист должен разместить значение аргумента в памяти, а в стек записать 32-битный указатель на него. Таким образом, все передаваемые функции API параметры представляются 32-битными величинами, и количество байт, занимаемых в стеке для передачи аргументов, кратно четырем;

- вызывающая программа загружает аргументы в стек последовательно, начиная с последнего, указанного в описании функции, и кончая

[1] Использованы материалы сайта wasm.ru и библиотека проекта assembler.ru.

первым. После загрузки всех аргументов программа вызывает функцию командой call;

- за возвращение стека в исходное состояние после возврата из функции API отвечает сама эта вызываемая функция. Программисту нет необходимости заботиться о восстановлении указателя стека ESP;
- вызываемая функция API гарантированно сохраняет регистры общего назначения EBP, ESI, EDI. Регистр EAX, как правило, содержит возвращаемое значение. Состояние остальных регистров после возврата из функции API следует считать неопределенным. (Полный набор соглашений stdcall регламентирует также сохранение системных регистров DS и SS. Однако для flat-модели памяти, используемой в win32, эти регистры значения не имеют).

На системном уровне этот набор соглашений дополняется способом формирования имени функции в объектном файле. Компилятор добавляет к началу имени функции (взятого из исходного текста программы) символ подчеркивания, а к концу - выражение вида @n, где n - десятичное число, равное количеству байт, занятому в стеке под аргументы функции. Так формируется технологическое имя, позволяющее осуществить связывание не только по имени вызываемой функции, но и по количеству ее аргументов. Благодаря ему при сборке обнаруживаются ошибки программиста в случае, когда он задал для вызываемой функции неправильное число аргументов.

Прежде чем переходить к рассмотрению примеров, следует обсудить еще одно обстоятельство. Оно связано с тем, что Windows поддерживает два типа кодировки текстовых символов - восьмибитную ANSI и шестнадцатибитную Unicode. По этой причине для всех функций API, так или иначе работающих с текстовыми символами, существует по два программных варианта. Тот, который работает с кодировкой ANSI, имеет суффикс A, например, CreateFileA. Тот, который работает с кодировкой Unicode, имеет суффикс W, например, CreateFileW.

Минимальная программа для ОС Windows

Задача написания минимальной работающей программы является базовой при ознакомлении с любым незнакомым окружением, будь то операционная система, среда для встраеваемых модулей или же нечто другое.

Минимальная программа обязана решать две задачи:

1. Корректно стартовать, получая при этом доступ к ресурсам рабочей среды;
2. Корректно завершаться, оставляя рабочую среду работоспособной.

Среда win32 выдвигает требование, согласно которому программа должна определять функцию WinMain, которая и являет точкой входа в программу. В технической документации на ОС функция WinMain объявляется следующим образом:

```
int WINAPI WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow
);
```

Параметры данной функции не имеют существенного значения для выполнения этой лабораторной работы и поэтому подробно описываться не будут. Желающие могут получить дополнительную информацию в соответствующей статье из базы знаний Microsoft (http://msdn2.microsoft.com/en-us/library/ms633559.aspx).

Таким образом, минимальная рабочая программа для win32 (написанная для компилятора MASM) записывается следующим образом:

```
.386                    ;(1)
.model flat,stdcall     ;(2)
ExitProcess PROTO :DWORD ;(3)
.code                   ;(4)
WinMain PROC PUBLIC hinst,prev_hinst,command_line,cmd_show ;(5)
;...                    ;(6)
INVOKE ExitProcess,0    ;(7)
WinMain ENDP            ;(8)
end                     ;(9)
```

1. .386 - директива ассемблера, определяющая набор команд процессора, который может быть использован в программе. Для приложений win32 необходимо использовать эту директиву или выше, в зависимости от того, собираетесь ли вы использовать возможности, предоставляемые процессорами последующих поколений;

2. .model flat,stdcall - сегментная директива ассемблера, определяющая сегментную модель приложения как плоскую, использующую соглашения о вызове процедур, принятые в win32. Именно такая сегментная модель должна всегда использоваться при написании приложений для win32;

3. ExitProcess PROTO :DWORD - прототип функции API, выполняющей завершение приложения. (Поскольку сервис этой функции предоставляется dll-библиотекой kernel32.dll, то *при сборке приложения необходимо подключить библиотеку импорта kernel32.lib*);

4. .code - сегментная директива ассемблера, определяющая начало сегмента кода;

5. WinMain PROC PUBLIC hinst,prev_hinst,command_line,cmd_show - начало тела стартовой процедуры. Следует обратить внимание на наличие директивы видимости (visibility) PUBLIC, которая позволит

компоновщику сделать процедуру доступной для операционной системы, дабы та смогла передать управление приложению;

6. ;... - здесь должен размещаться основной код приложения;

7. INVOKE ExitProcess, 0 - вызов функции завершения приложения. В данном случае используется код выхода 0, но он может быть любым в пределах 32-разрядного целого. В полноценных приложениях нормой считается определение кода выхода в цикле обработки сообщений главного окна;

8. * WinMain ENDP - директива, определяющая конец тела функции WinMain. В силу наличия предыдущей строки, он в данной программе не достижим, но это и не требуется: определения конца тела является синтаксическим требованием транслятора;

9. end - конец модуля.

Следует обратить особое внимание на строки 3 и 7. В них используются специальный расширения MASM, предназначенные для облегчения процесса вызова функций. Использование PROTO/invoke позволяет избавиться от вызова функций при помощи ручного заполнения стека параметрами и помогает избежать ошибок, связанных с неверным числом параметров. Более подробно про эти расширения рассказывается на странице базы знаний Microsoft http://support.microsoft.com/kb/73407.

Порядок выполнения работы

1. Перенести программу из л/р №3 под окружение win32;

2. Найти в библиотеке MSDN (http://msdn.microsoft.com) или ином справочнике по WinApi функции, выполняющие (1) преобразование числа в его строковое представление и (2) вывод строки при помощи стандартного диалогового окна;

3. Реализовать вывод результата вычислений при помощи найденных функций.