Лекция №3. ПРАКТИЧЕСКИЕ РЕКОМЕНДАЦИИ И МЕТОДЫ УСТАНОВКИ

Драйверы устройств

Контрольный список, используемый при разработке драйверов. Это перечень положений, что нужно учитывать при написании текста драйвера.

1. Для тестирования драйверов всегда используется тестовый диск;

2. начинается программа драйвера с адреса 0.

3. Драйвер написан в COM формате.

4. правильная структура данных заголовка запроса.

5. записан в поле связи заголовка драйвера код 1.

6. Правильно установленные биты в слове атрибутов заголовке драйвера ?.

7. Имеет ли основная процедура тип Far.

8. правильные директивы Assume.

9. выполнена во всех случаях замена сегмента по умолчанию на сегмент CS.

10. верный содержание регистров ES и BX, когда устанавливается слово состояния.

11. сохраняют локальные процедуры используемые в них регистры.

12. сохраняют свои значения локальные регистры по возвращении из локальной процедуры или обработки прерывания;

13. восстанавливаются все регистры хранившихся. Рассмотрим

каждый пункте отдельно.

Первый пункт. Для того, чтобы облегчить тестирование, нужно создать тестовый диск, который будет использоваться при загрузке системы с новыми драйверами устройств. Это позволяет локализовать процесс тестирования и избежать возможного воздействия на операционную среду. Для разных версий системы нужно использовать различные диски. Это обеспечит гарантию верности работы драйвера во всех версиях системы. Эту же тестовую дискету можно использовать для проверки работы драйвера на другом компьютере.

второй пункт связан с различием между обычными программами на языке ASM и программами драйверов. Обычные программы начинают выполняться с адреса 100h. Для драйверов директива org 100h не допускается.

третий пункт несмотря на то, что начиная с версий 3.0 системы - поддерживает работу драйверов, имеют формат .EXE, лучше создавать драйвера в формате

.COM. Для преобразования .EXE формата на .COM формат используется утилита EXE2BIN.

пункт 4 контролирует правильность обработки данных, система посылает драйверу. Можно избежать многих ошибок, используя структуры.

пункт 6 часто не учитывают, особенно при модификации существующего драйвера. Установка битов важна для определения типа устройства системы. Если не установить все необходимые биты, то некоторые команды не будут выполняться.

пункт 7 . При вызове драйвера используется команда дальнего вызова, поэтому в драйвере должно использоваться возвращения того же типа. Если тип far не упомянуть, возникает много проблем, особенно со стеком и указателем команд.

Пункты 8 и 9 . Драйверы используют только 1 сегмент, соответствующий требованиям к формату .COM-файлов. Сегментные регистры CS, DS, ES должны указывать на этот сегмент, объявляется с помощью директивы ASSUME. Другие сегменты в драйверах неприпускаються.

Другим важным обстоятельством является особенность передачи управления от системы к драйверу. Нельзя рассчитывать, что регистры DS и ES обеспечивают доступ к вашим переменных. Можно полагаться только на правильную установку регистра CS.

Поэтому все ссылки в драйвере на переменные должны сопровождаться заменой сегментного регистра по умолчанию на регистр CS.:MOV ES, CS: rh_seg.

Можно использовать прием, позволяющий избегать необходимости указывать сегментный префикс. В процедуре прерывания можно правильно установить содержание регистра так:

```
push CS сохранить CS
pop DS, установить в DS то же значение что и в CS Этот
прием также экономит память.
```

пункт 10 . Адрес заголовке запроса находится в регистрах ES и BX. Драйверы устройств используют эти два регистра при установлении слова состояния.

Поскольку процедура инициализации сохраняет все регистры в стеке, все они могут быть использованы в драйвере. Их надо хранить только перед вызовом процедур и использованием прерываний BIOS.

пункт 11 . Локальные процедуры, которые входят в состав драйвера, должны хранить используемые ими регистры. Лучший способ избежать ошибки -

документировать локальные процедуры, есть описывать регистры что используются, и регистры что возвращаются. Локальные процедуры должны сразу же после получения управления хранить все регистры используемых ими в стеке, а перед возвращением восстанавливать их из стека.

пункт 12 - логическое продолжение пункта 11.

В программах обработки команд обычно используются локальные процедуры или прерывания BIOS. При этом вызвана процедура может изменить значение регистров. Например, прерывание BIOS 10h, меняет содержание регистров BP, SI, DI. Поэтому, если драйвер использует эти регистры, их необходимо сохранить перед использованием этого прерывания.

пункт 13 . Нужно быть осторожными при разработке локальных процедур.

Начинать процедуру нужно с определения ее директивами PROC и ENDP. Затем писать ее текст. Перед использованием регистра для выполнения вычислений, нужно сохранить его содержание в стеке. В конце процедуры - восстановить регистр. Если в стеке записано несколько значений, восстанавливать их надо в порядке, обратном записи.

Лучший способ разработки драйвера заключается в том, чтобы оформить сначала процедуры обработки команд в виде обычной программы на языке ассемблера. Такой подход называется разработкой прототипа. Использование такого метода позволяет проверить работоспособность каждой команды перед тем, как включить ее в драйвер.

Прототипы использовать целесообразно, потому что:

1. Для загрузки драйвера в память и его трассировки нельзя использовать утилиту DEBUG. (Драйверы - это часть системы и загружаются в память до того, как можно выполнить утилиту DEBUG.

2. Можно включать фрагменты обработки команд в драйвер по мере того, как они проходят процедуру отладки в обычной программе.

Такой метод позволяет создавать драйвер поэтапно. Главным препятствием является процедура выполнения команды инициализации. Процесс ее выполнения невозможно проконтролировать.

Команда инициализации используется для подготовки драйвера к вызовам из программ. По этой команде можно вывести сообщение, инициализировать устройство, установить векторы прерываний и добавить резидентную программу. если процедура

выполнения команды инициализации образуется очень сложной, лучше выполнить те же операции, используя IOCTL - последовательности.

В прототипе драйвера устройства можно использовать все структуры с драйвера. Туда можно включить переменные заголовке запроса и локальные переменные.

Другой метод отладки драйверов основан на запоминании в переменных значений при их изменении в драйвере. Любое изменение до критических значений записывается в этих переменных. Для их просмотра можно использовать утилиту DEBUG. Хотя невозможно изменить последовательность выполнения команд или определить точку прекращения, можно увидеть, принимают переменные правильные значения.

Для того, чтобы определить место в памяти, куда загружен драйвер, нужно вывести на экран адрес драйвера в процессе выполнения команды инициализации:

```
push cs                         Сохранить сегм. адр.
pop dx                          ; В DX
lea di, pmsgla                          ; Адрес строке ASCII
call hex2asc                            ; Преобраз. DX на строку
mov ah, 9                       ; Вывода строке
lea dx, pmsgl                           , На экран
int 21h
pmsgl db 'Драйвер устройства загружен по адресу "
pmsgla db '0000: 0000h ", 0Dh, 0Ah,' $ '
```

Эта последовательность команд выводит на экран сегментный адрес драйвера, используя функцию системы. Процедура HEX2ASC превратит 16-битную адрес в ASCII-последовательность.

Новый стек.

Драйверы обычно используют стек, в котором м.б. записано около 20 слов. Этого может оказаться недостаточно, если драйвер содержит много вызовов процедур. Использование стека позволяет сохранять регистры при входе в драйвер и позволяет вызвать из него другие процедуры. Любая из этих операций может привести к переполнению стека. В этом случае потребуется стек большего размера. Нужно определить новый стек. Для этого сначала нужно сохранить регистр сегмента стека SP в переменных, определенных в драйвере. Затем - установить

сегмент стека и указать на стек внутри драйвера. Стек может представлять собой массив байтов.

Процедуры, позволяющие драйвера заменить стек системы на собственный стек.

```
; Определение области данных для
; Нового стека и хранения регистров
; Старого стека.
stack_ptr dw?                    ; Указатель старого стека.
stack_seg dw?                    ; Сегмент старого стека.
newstack db 100h dup (?)
newstacktop equ $ -2             , Размер нового стека.
; Переключение на новый стек.
switch2new: proc near            ; Переключить на новый стек.
cli                              ; Запретить прерывания.
mov cs: stack_ptr, sp            сохранить
mov cs: stack_seg, ss            ; SS и SP
mov ax, cs                       ; Получить текущий. сегмент
mov ss, ax                       ; Уст. сегмент стека
mov sp, newstacktop              ; Установить указатель стека
sti; позволить переривания
ret
switch2new endp
; Переключение на старый стек
switch2old: proc near
cli
mov ss, cs: stack_seg
mov sp, cs: stack_ptr
sti
ret
switch 2old endp
```

Новый стек определяется при входе в драйвер, путем вызова в процедуре прерывания процедуры switch2new (установление нового стека) сразу после команд, сохраняющие регистры старого стека. Затем, перед выходом из драйвера, нужно восстановить старый стек. Процедуру switch2old (восстановление стека) следует вызвать после установки бита *ВЫПОЛНЕНО* слова состояния заголовке запроса в секции общего выхода, но перед восстановлением регистров с старого стека.

При переключении стеков прерывания запрещаются. Это предотвращает нарушение процесса изменения стека в случае поступления прерываний. Длина нового стека устанавливается в зависимости от потребностей драйвера.

Особый бит.

Особый бит (4) слова атрибутов заголовке драйвера показывает, что драйвер терминала поддерживает быстрое средство вызова символов. Этот бит анализируется только драйверами консоли. Если он установлен, то драйвер должен записать в вектор прерывания 29h адрес процедуры быстрого ввода символов.

Обычно система проверяет каждый символ вводится с клавиатуры, для того, чтобы выявить Ctrl-C. Такой режим называется ASCII-режимом. Когда нужно организовать быстрый вывод на терминал, поток символов не проверяется, что называется двоичным режимом. Программы реализуют вывода в двоичном режиме

с помощью IOCTL-функций системы управления в / в (44h) при установленном бите 5 в регистре DX.

Поскольку за установление прерывания 29h соответствует драйвер, то в любом драйвере должен устанавливаться только один особый бит.

```
; Последовательность команд, обеспечивающих быстрый обмен с консолью.
int29h: sti
push ax
push bx
mov bl, 07h                    ; Белые символы на черном фоне
mov ah, 0Eh
int 10h
pop bx
pop ax
; Инициализировать вектор прерывания 29h адресу процедуры, int29h
set29h: mov bx, 0a4h                    , Адрес 29h
lea ax, int29h                                  ; Смещение процедуры int29h
mov [bx], ax                    ; Установить смещение в векторе перер.29h
mov [bx + 2], cs                ; сегмент
```

Эту последовательность команд необходимо использовать только в том случае, если в слове атрибутов заголовке драйвера установлен особый бит 4. Эти команды добавляются к процедуре выполнения команды инициализации.

Написание драйверів на языке С / С ++.

Написание драйверов на языке Си. Особенности написания драйверов на языках высокого уровня.

Первой особенностью драйвера является то, что его заголовок должен размещаться в самом начале программы и первым загружаться в память. Кроме того, при загрузке драйверов не требуется выделять область памяти " памяти под PSP.

В этой " связи с этим возникают 2 проблемы. В программах, написанных на языках высокого уровня, нельзя управлять загрузкой программы в память. Как и куда будут загружаться программные строки, данные, стек и куча, устанавливается в процессе компоновки. То есть, нужно найти средство, позволяющее загружать заголовок на начало файла драйвера.

Вторая проблема связана с тем, что компиляторы Си по змовчанню генерируют несколько программных сегментов и сегментов данных. Для программы, данных и стека выделяются различные сегменты. Но нужно, чтобы создавался только один сегмент.

Особенности языка Си.

Для обращения к данным, хранящимся вне драйвером, используются указатели. С их помощью осуществляется обмен данными с системой. Для определения заголовков запроса, специфичных для каждой команды, используются структуры. Ссылка на структуры и их элементы осуществляются с помощью указателей. Для драйверов используются крошечная (tini) и малая (small) модели памяти. Программа и данные размищуються в одном сегменте, поэтому для них используются ближние посылки.

Тип посылок зависит не только от модели памяти, но и от того, где в памяти хранятся данные. Таким образом, при чтении и записи данных необходимо проверять, какой тип ссылок следует использовать: дальний или ближний.

Ограничения, установленные при программировании на Си.

Первое из ограничений, которое связано с использованием библиотечных функций Си, большинство из которых включают вызовы системы. При вызове с драйвера такой функции произойдет сбой, потому что система не является повторно вводимого.

Второе ограничение связанно с использованием стека с Си. Когда происходит передача управления драйвера, то активизируется по змовчанню программный стек, который

имеет недостаточный объем. Его пространства хватает только для выполнения 20-ти операций PUSH. Кроме того, в Си-программах стек используется для хранения локальных переменных для каждой процедуры. Таким образом, области стека может не хватать.

Есть два решения этой проблемы:

1. Использовать стек как реже. Переменные объявляются глобальными, по счет чего отпадает необходимость передачи данных при вызовах функций и использовании локальных переменных. Однако, использование большого числа глобальных переменных считается плохим стилем.

2. Лучшее решение состоит в том, чтобы увеличивать размер стека внутри драйвера перед вызовом любых Си-процедур. Это даст возможность управлять объемом стеке.