ЛЕКЦІЯ №3. ПРАКТИЧНІ РЕКОМЕНДАЦІЇ І МЕТОДИ НАЛАШТУВАННЯ ДРАЙВЕРІВ ПРИСТРОЇВ

Контрольний список, використовуваний при розробці драйверів. Це перелік положень, що потрібно враховувати при написанні тексту драйвера.

1. Для тестування драйверів завжди використовується тестовий диск;
2. Чи починається програма драйвера з адреси 0.
3. Драйвер написаний в COM форматі.
4. Чи правильна структура даних заголовка запиту.
5. Чи записаний у поле зв'язку заголовка драйвера код -1.
6. Чи правильно встановлені біти в слові атрибутів заголовку драйвера?.
7. Чи має основна процедура тип Far.
8. Чи правильні директиви Assume.
9. Чи виконана в усіх випадках заміна сегменту по замовчанню на сегмент CS.
10. Чи вірний вміст регістрів ES і BX, коли встановлюється слово стану.
11. Чи зберігають локальні процедури використовувані в них регістри.
12. Чи зберігають свої значення локальні регістри після повернення з локальної процедури або обробки переривання;
13. Чи відновлюються всі регістри що зберігалися.

Розглянемо кожний пункти окремо.

Перший пункт. Для того, щоб полегшити тестування, потрібно створити тестовий диск, що буде використовуватися при завантаженні системи із новими драйверами пристроїв. Це дозволяє локалізувати процес тестування й уникнути можливої дії на операційне середовище. Для різних версій системи потрібно використовувати різні диски. Це забезпечить гарантію вірності роботи драйвера в усіх версіях системи. Цю ж тестову дискету можна використовувати для перевірки роботи драйвера на іншому комп'ютері.

Другий пункт пов'язаний з відмінністю між звичайними програмами на мові ASM і програмами драйверів. Звичайні програми починають виконуватися з адреси 100h. Для драйверів директива org 100h не припустима.

Третій пункт незважаючи на те, що починаючи з версій 3.0 системи - підтримує роботу драйверів, що мають формат .EXE, краще створювати драйвера у форматі

.COM. Для перетворення .EXE формату на .COM формат використовується утиліта EXE2BIN.

Пункт 4 контролює вірність обробки даних, які система посилає драйверу. Можна уникнути багатьох помилок, використовуючи структури.

Пункт 6 часто не враховують, особливо при модифікації існуючого драйвера.

Встановлення бітів важлива для визначення типу пристрою для системи. Якщо не встановити всі потрібні біти, то деякі команди не будуть виконуватися.

Пункт 7. При виклику драйвера використовується команда дальнього виклику, тому в драйвері повинне використовуватися повернення того ж типу. Якщо тип far не зазначити, виникає багато проблем, особливо зі стеком і покажчиком команд.

Пункти 8 і 9. Драйвери використовують тільки 1 сегмент, що відповідає вимогам до формату .COM-файлів. Сегментні регістри CS, DS, ES повинні вказувати на цей сегмент, що оголошується за допомогою директиви ASSUME. Інші сегменти в драйверах неприпускаються.

Іншою важливою обставиною є особливість передачі керування від системи до драйвера. Не можна розраховувати, що регістри DS і ES забезпечують доступ до ваших змінних. Можна покладатися тільки на правильну установку регістру CS.

Тому всі посилання в драйвері на змінні повинні супроводжуватися заміною сегментного регістру по замовчанню на регістр CS.:MOV ES,CS:rh_seg.

Можна використовувати прийом, що дозволяє уникати необхідності вказувати сегментний префікс. У процедурі переривання можна правильно установити вміст регістру так:

```
push CS ;зберегти CS
pop DS ;встановити в DS те ж значення що й у CS
```

Цей прийом також заощаджує пам'ять.

Пункт 10. Адреса заголовку запиту знаходиться в регістрах ES і BX. Драйвери пристроїв використовують ці два регістри при встановленні слова стану.

Оскільки процедура ініціалізації зберігає всі регістри в стеку, усі вони можуть бути використані в драйвері. Їх треба зберігати тільки перед викликом процедур і використанням переривань BIOS.

Пункт 11. Локальні процедури, які входять до складу драйвера, повинні зберігати використовувані ними регістри. Кращий засіб уникнути помилки -

документувати локальні процедури, тобто описувати регістри що використовуються, та регістри що повертаються. Локальні процедури повинні відразу ж після одержання керування зберігати всі регістри що ними використовуються в стеку, а перед поверненням відновлювати їх зі стеку.

Пункт 12 - логічне продовження пункту 11.

У програмах обробки команд звичайно використовуються локальні процедури або переривання BIOS. При цьому викликана процедура може змінити значення регістрів. Наприклад, переривання BIOS 10h, змінює вміст регістрів BP, SI, DI. Тому, якщо драйвер використовує ці регістри, їх необхідно зберегти перед використанням цього переривання.

Пункт 13. Потрібно бути обережними при розробці локальних процедур. Починати процедуру потрібно з визначення її директивами PROC і ENDP. Потім - писати її текст. Перед використанням регістру для виконання обчислень, потрібно зберегти його вміст у стеку. Наприкінці процедури - відновити регістр. Якщо в стеку записано декілька значень, відновлювати їх треба в порядку, зворотному запису.

Найкращий засіб розробки драйвера полягає в тому, щоб оформити спочатку процедури обробки команд у вигляді звичайної програми на мові асемблера. Такий підхід називається розробкою прототипу. Використання такого методу дозволяє перевірити працездатність кожної команди перед тим, як включити її в драйвер.

Прототипи використовувати доцільно, тому що:

1. Для завантаження драйвера в пам'ять і його трасування не можна використовувати утиліту DEBUG. (Драйвери - це частина системи і завантажуються в пам'ять до того, як можна виконати утиліту DEBUG.

2. Можна включати фрагменти обробки команд у драйвер по мірі того, як вони проходять процедуру налагодження в звичайній програмі.

Такий метод дозволяє створювати драйвер поетапно. Головною перешкодою є процедура виконання команди ініціалізації. Процес її виконання неможливо проконтролювати.

Команда ініціалізації використовується для підготовки драйвера до викликів з програм. По цій команді можна вивести повідомлення, ініціалізувати пристрій, установити вектори переривань і додати резидентну програму. Якщо процедура

виконання команди ініціалізації утворюється дуже складною, краще виконати ті ж операції, використовуючи IOCTL - послідовності.

У прототипі драйвера пристрою можна використовувати всі структури з драйвера. Туди можна включити змінні заголовку запиту і локальні змінні.

Інший метод налагодження драйверів заснований на запам'ятовуванні у змінних значень при їхній зміні в драйвері. Будь-яка зміна до критичних значень записується в цих змінних. Для їх перегляду можна використовувати утиліту DEBUG. Хоча неможливо змінити послідовність виконання команд або визначити точку припинення, можна побачити, чи приймають змінні правильні значення.

Для того, щоб визначити місце в пам'яті, куди завантажений драйвер, потрібно вивести на екран адресу драйвера в процесі виконання команди ініціалізації:

```
push cs             ;зберегти сегм. адр.
pop dx              ;у DX
lea di,pmsgla             ;адреса рядку ASCII
call hex2asc              ;перетвор. DX на рядок
mov ah,9            ;виведення рядку
lea dx,pmsgl              ;на екран
int 21h
pmsgl db 'Драйвер пристрою завантажений за адресою '
pmsgla db '0000:0000h',0Dh,0Ah,'$'
```

Ця послідовність команд виводить на екран сегментну адресу драйвера, використовуючи функцію системи. Процедура HEX2ASC перетворить 16-бітну адресу в ASCII-послідовність.

Новий стек.

Драйвери звичайно використовують стек, у якому м.б. записано близько 20 слів. Цього може виявитися недостатньо, якщо драйвер містить багато викликів процедур. Використання стеку дає можливість зберігати регістри при вході в драйвер і дозволяє викликати з нього інші процедури. Будь-яка з цих операцій може призвести до переповнення стеку. У цьому випадку буде потрібний стек більшого розміру. Потрібно визначити новий стек. Для цього спочатку потрібно зберегти регістр сегменту стеку SP у змінних, які визначені в драйвері. Потім - встановити

сегмент стеку і вказати на стек усередині драйвера. Стек може являти собою масив байтів.

Процедури, що дозволяють драйверу замінити стек системи на власний стек.

```
;Визначення області даних для
;нового стеку і зберігання регістрів
;старого стеку.
stack_ptr dw ?           ;покажчик старого стеку.
stack_seg dw ?           ;сегмент старого стеку.
newstack db 100h dup (?)
newstacktop equ $-2      ;розмір нового стеку.
;Переключення на новий стек.
switch2new: proc near    ;переключити на новий стек.
cli                      ;заборонити переривання.
mov cs:stack_ptr,sp      ;зберегти
mov cs:stack_seg,ss      ;SS i SP
mov ax,cs                ;одержати поточний. сегмент
mov ss,ax                ;уст. сегмент стеку
mov sp,newstacktop       ;установити покажчик стеку
sti ;дозволити переривания
ret
switch2new endp
;Переключення на старий стек
switch2old: proc near
cli
mov ss,cs:stack_seg
mov sp,cs:stack_ptr
sti
ret
switch 2old endp
```

Новий стек визначається при вході в драйвер, шляхом виклику в процедурі переривання процедури switch2new (встановлення нового стеку) відразу після команд, що зберігають регістри старого стеку. Потім, перед виходом із драйвера, потрібно відновити старий стек. Процедуру switch2old (відновлення стеку) варто викликати після встановлення біту *ВИКОНАНО* слова стану заголовку запиту в секції загального виходу, але перед відновленням регістрів з старого стеку.

При переключенні стеків переривання забороняються. Це запобігає порушенню процесу зміни стеку у випадку надходження переривань. Довжина нового стеку встановлюється в залежності від потреб драйверу.

Особливий біт.

Особливий біт (4) слова атрибутів заголовку драйвера показує, що драйвер терміналу підтримує швидкий засіб виклику символів. Цей біт аналізується тільки драйверами консолі. Якщо він встановлений, то драйвер повинен записати у вектор переривання 29h адресу процедури швидкого введення символів.

Зазвичай система перевіряє кожен символ, який вводиться з клавіатури, для того, щоб виявити Ctrl-C. Такий режим називається ASCII-режимом. Коли потрібно організувати швидке виведення на термінал, потік символів не перевіряється, що називається двійковим режимом. Програми реалізують виведення у двійковому режимі за допомогою IOCTL-функцій системи керування в/в (44h) при встановленому біті 5 у регістрі DX.

Оскільки за встановлення переривання 29h відповідає драйвер, то в будь-якому драйвері повинен встановлюватися тільки один особливий біт.

```
;Послідовність команд, що забезпечують швидкий обмін із консоллю.
int29h: sti
push ax
push bx
mov bl,07h          ;білі символи на чорному фоні
mov ah,0Eh
int 10h
pop bx
pop ax
;Ініціалізувати вектор переривання 29h адресою процедури, int29h
set29h: mov bx,0a4h       ;адреса 29h
lea ax,int29h                    ;зсув процедури int29h
mov [bx],ax              ;установити зсув у векторі перер.29h
mov [bx+2],cs            ;сегмент
```

Цю послідовність команд необхідно використовувати тільки в тому випадку, якщо в слові атрибутів заголовку драйвера встановлений особливий біт 4. Ці команди добавляються до процедури виконання команди ініціалізації.

Написання драйверів на мові C/C++.

Написання драйверів на мові Сі. Особливості написання драйверів на мовах високого рівня.

Першою особливістю драйвера є те, що його заголовок повинен розміщатися на самому початку програми і першим завантажуватися в пам'ять. Крім того, при завантаженні драйверів не потрібно виділяти область пам′яті під PSP.

У зв′язку з цим виникають 2 проблеми. У програмах, написаних на мовах високого рівня, не можна управляти завантаженням програми в пам'ять. Як і куди будуть завантажуватися програмні рядки, дані, стек і купа, встановлюється в процесі компоновки. Тобто, потрібно знайти засіб, що дозволяє завантажувати заголовок на початок файлу драйвера.

Друга проблема пов'язана з тим, що компілятори Сі по змовчанню генерують декілька програмних сегментів і сегментів даних. Для програми, даних і стеку виділяються різні сегменти. Але потрібно, щоб створювався тільки один сегмент.

Особливості мови Сі.

Для звернення до даних, що зберігається поза драйвером, використовуються покажчики. З їхньою допомогою здійснюється обмін даними з системою. Для визначення заголовків запиту, специфічних для кожної команди, використовуються структури. Посилання на структури і їхні елементи здійснюються за допомогою покажчиків. Для драйверів використовуються малюсінька (tini) і мала (small) моделі пам'яті. Програма і дані розміщуються в одному сегменті, тому для них використовуються ближні посилки.

Тип посилок залежить не тільки від моделі пам'яті, але і від того, де в пам'яті зберігаються дані. Таким чином, при читанні та запису даних необхідно перевіряти, який тип посилань варто використовувати: дальній або ближній.

Обмеження, які накладаються при програмуванні на Сі.

Перше з обмежень, яке пов’язано з використанням бібліотечних функцій Сі, більшість з яких включають виклики системи. При виклику з драйвера такої функції відбудеться збій, тому що система не є повторно вводимою.

Друге обмеження пов’язанно з використанням стеку із Сі. Коли відбувається передача керування драйверу, то активізується по змовчанню програмний стек, який

має недостатній обсяг. Його простору вистачає тільки для виконання 20-ти операцій PUSH. Крім того, у Сі-програмах стек використовується для збереження локальних змінних для кожної процедури. Таким чином, області стеку може не вистачати.

Є два вирішення цієї проблеми:

1. Використовувати стек як найрідше. Змінні оголошуються глобальними, за рахунок чого відпадає необхідність передачі даних при викликах функцій і використанні локальних змінних. Проте, використання великого числа глобальних змінних вважається поганим стилем.

2. Краще рішення полягає в тому, щоб збільшувати розмір стеку всередині драйвера перед викликом будь-яких Сі-процедур. Це дасть можливість управляти обсягом стеку.