МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Операционные системы»

Тема: Построение модуля оверлейной структуры

Студент гр. 8381 _____ Почаев Н.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Исследование возможности построение загрузочного модуля оверлейной структуры. Исследуется структура оверлейного сегмента и способ загрузки и выполнения оверлейных сегментов. Для запуска вызываемого оверлейного модуля используется функция 4B03h прерывания int 21h. Все загруженные и оверлейные модули находятся в одном каталоге.

В этой работе также рассматривается приложение, состоящее из нескольких модулей, поэтому все модули помещаются в один каталог и вызываются с использованием полного пути.

Основные теоретические положения.

Для организации программы, имеющей оверлейную структуру, используется функция 4B03h прерывания int 2lh. Эта функция позволяет в отведенную область памяти, начинающуюся с адреса сегмента, загрузить программу, находящуюся в файле на диске.

Передача управления загруженной программе этой функцией не осуществляется и префикс сегмента программы (PSP) не создается. Обращение к функции 4B03h:

```
AX=4B03h - код функции;
DS:DX - указывает на строку ASCIIZ, содержащую путь к оверлею;
ES:BX - указатель на блок параметровг который представляет собой два слова
памяти, содержащих сегментный адрес загрузки программы.
```

Если флаг переноса CF=1 после выполнения функции, то произошли ошибки и регистр AX содержит код ошибки. Значение регистра AX характеризует следующие ситуации:

- 1 - несуществующая функция;
- 2 - файл не найден;
- 3 - маршрут не найден;
- 4 - слишком много открытых файлов;
- 5 - нет доступа;

- 8 - мало памяти;
- 10 - неправильная среда.

Если флаг переноса CF=0, то оверлей загружен в память.

Перед загрузкой оверлея вызывающая программа должна освободить память по функции 4Ah прерывания int 2 lh. Затем определить размер оверлея. Это можно сделать с помощью функции 4Eh прерывания 21h. Перед обращением к функции необходимо определить область памяти размером в 43 байта под буфер DTA, которую функция заполнит, если файл будет найден.

Функция использует следующие параметры: CX - значение байта атрибутов, которое для файла имеет значение 0; DS:DX - указатель на путь к файлу, который записывается в формате строки ASCIIZ.

Если флаг переноса CF=1 после выполнения функции, то произошли ошибки и регистр AX содержит код ошибки. Значение регистра AX характеризует следующие ситуации:

- 2 - файл не найден;
- 3 - маршрут не найден.

Если CF=0, то в области памяти буфера DTA со смещением lAh будет находится младшее слово размера файла, а в слове со смещением ICh - старшее слово размера памяти в байтах.

Полученный размер файла следует перевести в параграфы, причем следует взять большее целое числа параграфов. Затем необходимо отвести память с помощью функции 48h прерывания 21h. После этого необходимо сформировать параметры для функции 4B03h и выполнить ее.

После отработки оверлея необходимо освободить память с помощью функции 49h прерывания int 21 h. Обращение к этой функции содержит следующие параметры:

```
AH=49h - код функции;
ES - сегментный адрес освобождаемой памяти.
```

Оверлейный сегмент не является загрузочным модулем типов .COM или .EXE. Он представляет собой кодовый сегмент, который оформляется в ассемблере как функция с точкой входа по адресу 0 и возврат осуществляется командой RETF. Это необходимо сделать, потому что возврат управления должен быть осуществлен в программу, выполняющую оверлейный сегмент. Если использовать функции выхода 4Ch прерывания int 21 h, то программа закончит свою работу.

Выполнение работы.

Выполнение работы производилось на базе операционной системы Windows XP (32 bit), запускаемой в системе виртуализации VMware Workstation, в редакторе Notepad++. Сборка и отладка модулей производились с помощью компилятора MASM и отладчика AFD. Также в работе был использован консольный файловый менеджер Far Manager и HEX-редактор HxD. Для дополнительного тестирования и проверки функциональности программы использовался DOSBox.

Ход работы.

Последовательность действий, выполняемых программой:

1. При запуске программы освобождается место для загрузки оверлеев.
2. Определяем размер файла оверлея.
3. Выделяем место для загрузки оверлея.
4. Загружаем оверлей.
5. Если файл оверлея не найден, выводим соответствующее сообщение.
6. Если найден, то обращаемся к сегменту оверлея.
7. Оверлей выводит свой адрес (адрес сегмента) в 16-чном виде.
8. Возвращаемся в вызывающую программу.
9. Освобождаем память, занятую оверлеем.

10. Проверяем, все ли оверлеи загружены. Если да, то выход, иначе загружаем дальше.

На рис. 1 приведен вывод при загрузке оверлеев.

```
E:\Labs\lr7>LR7.EXE

WAY: E:\LABS\LR7\ovr1.ovl
Overlay segment address: 059F

WAY: E:\LABS\LR7\ovr2.ovl
Overlay segment address: 059F
```

Рисунок 1 – Загрузка программы и вывод адресов оверлеев

Аналогичный результат будет получен в случае запуска программы из другого каталоге, что отображено на рис. 2.

```
E:\Labs\test_7>LR7.EXE

WAY: E:\LABS\TEST_7\ovr1.ovl
Overlay segment address: 059F

WAY: E:\LABS\TEST_7\ovr2.ovl
Overlay segment address: 059F
```

Рисунок 2 – Запуск программы из другого каталога

Результат работы программы, в случае, если будет убран 2-ой оверлей представлен на рис. 3.

```
E:\Labs\test_7>LR7.EXE

WAY: E:\LABS\TEST_7\ovr1.ovl
Overlay segment address: 059F

WAY: E:\LABS\TEST_7\ovr2.ovl
Error! File not found.
```

Рисунок 3 – Аварийное завершение программы

Выводы.

В ходе данной работы были исследованы возможности построения загрузочного модуля оверлейной структуры. Рассмотрено приложение, состоящее из нескольких модулей. Реализовано приложение в возможностью запуска модуля оверлейной структуры из любого каталога.

Ответы на контрольные вопросы.

1. Как должна быть устроена программа, если в качестве оверлейного сегмента использовать .COM модули?

При использовании в качестве оверлейного сегмента .COM модуля, необходимо вызывать его по смещению 100h, поместив PSP в начале выделенной памяти, так как в .COM файлах код располагается с адреса 100h, в ином случае PSP запускаемого оверлея сформирован не будет.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
 data segment
    PSP                 dw    ?
    path          db 100 dup (?)
    dta           db 43 dup (?)
    overlay       dw 0
    epb           dw ?
    _ss           dw ?
    _sp           dw ?
    Count         db 0
    WAY           db 'WAY: $'
    Adress        db 'Overlay segment address: $'
    Error1        db 'Error! File not found.$'
    Error2        db 'Error! Path not found.$'
    file_ov1      db 'ovr1.ovl',0
    file_ov2      db 'ovr2.ovl',0
    endl    db 13, 10, '$'
  data ends

  stack segment
    dw 128 dup(0)
  stack ends

  code segment
    assume ds:data, ss:stack, cs:code, es:nothing
    .386

start:

      mov   ax, data
      mov   ds, ax
      mov   PSP, es

      mov         es, es:[002Ch]
      xor         bx, bx

;iioaai ai iooe
next:
      mov   dl, byte PTR es:[bx]
      cmp   dl, 0h
      je          first_0
      inc   bx
      jmp   next
first_0:
      inc   bx
```

```
      mov   dl, byte PTR es:[bx]
      cmp   dl, 0h
      je          second_0
      jmp   next

second_0:
      add         bx,3

      push  si
      mov         si, offset path
next1:
      mov   dl, byte PTR es:[bx]
      mov         [si], dl
      inc         si
      inc         bx
      cmp         dl, 0
      jne         next1

next2:
      mov         al, [si]
      cmp         al, '\'
      je          next3
      dec         si
      jmp         next2

next3:
      inc         si
      push  di
      mov         di, offset file_ov1
next4:
      mov         ah, [di]
      mov         [si], ah
      inc         si
      inc         di
      cmp         ah, 0
      jne         next4
      mov   ah,'$'
      mov   [si],ah
      pop         di

      mov         dx, offset endl
    mov     ah, 09h
    int     21h
      mov         dx, offset WAY
      mov   ah, 09h
    int     21h
      mov   dx, offset path
```

```
      mov   ah, 09h
    int     21h
      mov         dx, offset endl
    mov     ah, 09h
    int     21h

      mov   ax, PSP
      mov   es, ax
      mov   bx, offset last_byte
      shr   bx, 4
      add   bx, 50
      mov   ah, 4Ah
      int   21h

      mov   dx, offset dta
      mov   ah, 1Ah
      int   21h

AGAIN:
      mov   dx, offset path
      mov   ah, 4Eh
      mov   cx, 0h
      int   21h

jnc no_err
      cmp   ax, 2
      jne   err1
      mov   dx, offset Error1
      mov   ah, 09h
    int     21h
      mov         dx, offset endl
    mov     ah, 09h
    int     21h
      jmp         exit

err1:
      cmp   ax, 3
      jne   err2
      mov   dx, offset Error2
      mov   ah, 09h
    int     21h
      mov         dx, offset endl
    mov     ah, 09h
    int     21h
      jmp   exit

err2:
```

```
      cmp   ax, 18
      jne   exit
      mov   dx, offset Error1
      mov   ah, 09h
    int     21h
      mov         dx, offset endl
    mov     ah, 09h
    int     21h
      jmp   exit

no_err:
      mov   ebx, dword ptr [ offset dta + 1Ah ]
      shr   ebx, 4
      inc   ebx

      mov   ah, 48h
      int   21h
      mov   epb, ax
      mov   ax, ds
      mov   es, ax
      mov   bx, offset epb
      mov   dx, offset path
      mov   _sp, sp
      mov   _ss, ss

      mov   ax, 4B03h
      int   21h

      mov   ax, data
      mov   ds, ax
      mov   ss, _ss
      mov   sp, _sp

      mov   dx, offset Adress
      mov   ah, 09h
    int     21h

      push  ds
      call  dword ptr overlay
      mov         dx, offset endl
    mov     ah, 09h
    int     21h
      pop   ds

      mov   ax, epb
      mov   es, ax
      mov   ah, 49h
```

```
      int   21h

      mov   al, Count
      cmp   al, 1
      je          exit
      mov         dx, offset endl
    mov     ah, 09h
    int     21h
      mov   di, 0

next5:
      mov         al, [si]
      cmp         al, '\'
      je          next6
      dec         si
      jmp         next5

next6:
      inc         si

      push  di
      mov         di, offset file_ov2
next7:
      mov         ah, [di]
      mov         [si], ah
      inc         si
      inc         di
      cmp         ah, 0
      jne         next7
      mov   ah,'$'
      mov   [si],ah
      pop         di
      pop   si

      mov   Count, 1
      mov   dx, offset WAY
      mov   ah, 09h
    int     21h
      mov         dx, offset path
      mov   ah, 09h
    int     21h
      mov         dx, offset endl
    mov     ah, 09h
    int     21h

      jmp   AGAIN
```

```
exit:
      xor   al, al
      mov   ah, 4Ch
      int   21h
last_byte:
  code ends
    end start
```