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

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Операционные системы»

Тема: Построение модуля оверлейной структуры

Студентка гр. 0382 _____ Здобнова К.Д.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2022

Цель работы.

Исследование возможности построения 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 загрузочные 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.

Задание.

Шаг 1. Для выполнения лабораторной работы необходимо написать и отладить программный модуль типа .EXE, который выполняет функции:

Освобождает память для загрузки оверлеев.

Читает размер файлаоверлея и запрашивает объем памяти,достаточный для его загрузки.

Файл оверлейного сегмента загружается и выполняется.

Освобождается память, отведенная для оверлейного сегмента.

5) Затем действия 1)-4) выполняются для следующего оверлейногосегмента.

Шаг 2. Также необходимо написать и отладить оверлейные сегменты. Оверлейный сегмент выводит адрес сегмента, в который он загружен.

Шаг 3. Запустите отлаженное приложение. Оверлейные сегменты должны загружаться с одного и того же адреса, перекрывая друг друга.

Шаг 4. Запустите приложение из другого каталога. Приложение должно быть выполнено успешно.

Шаг 5. Запустите приложение в случае, когда одного оверлея нет в каталоге. Приложение должно закончиться аварийно.

Шаг 6. Занесите полученные результаты в виде скриншотов в отчет. Оформите отчет в соответствии с требованиями.

Выполнение работы.

Результат работы программы, когда оба оверлейных модуля в текущем

каталоге:

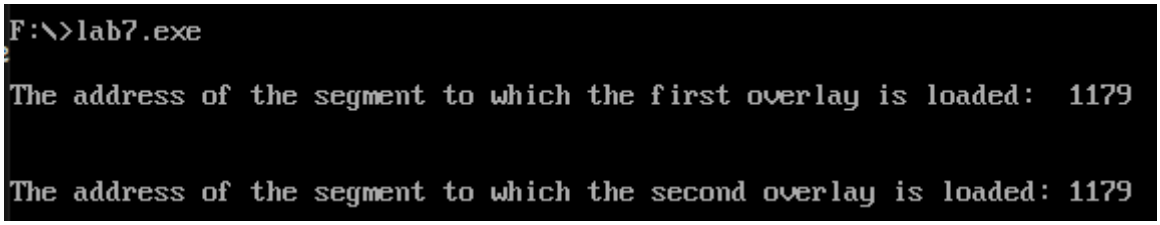

Рисунок 1.

Результат работы программы, когда оба оверлейных модуля не в текущем каталоге:

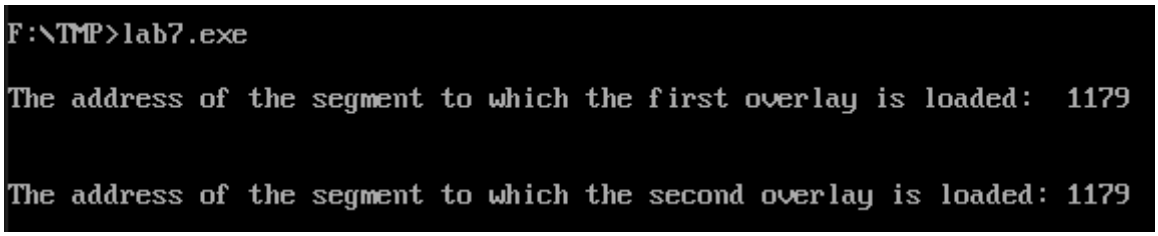

Рисунок 2.

Результат работы программы, когда один оверлейный модуль не в текущем каталоге:

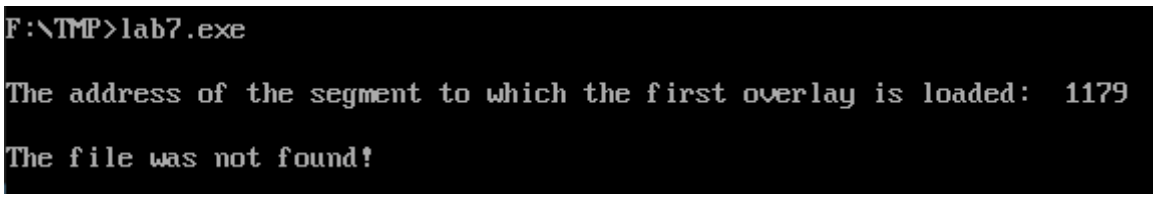

Рисунок 3.

Ответы на контрольные вопросы

1) Как должна выглядеть программа, если в качестве оверлейного сегмента использовать .COM модули?

После записи значений регистров в стек необходимо поместить регистр CS в регистр DS, так как адрес сегмента данных совпадает с адресом сегмента кода. Так же необходимо добавить 100h, т.к. изначально сегменты настроены на PSP.

Вывод

В ходе выполнения лабораторной работы был исследован принцип работы оверлейных структур и принцип их загрузки.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab7.asm

```
_CODE SEGMENT
     ASSUME CS:_CODE, DS:_DATA, ES:_DATA, SS:_STACK

_DATA SEGMENT
Mem_7                db     13,     10,'Memory     control     unit
destroyed',13,10,'$'
Mem_8                db  13,  10,'Not  enough  memory  to  perform  the
function',13,10,'$'
Mem_9                db   13,   10,'Wrong   address   of   the   memory
block',13,10,'$'
OVL_PATH            db 64  dup (0), '$'
DTA                 db 43 DUP (?)
KEEP_PSP            dw 0
ERROR_ALLOC                db  'Failed  to  allocate  memory  to  load
overlay!',13,10,'$'
BLOCK_ADDR          dw 0
CALL_ADDR           dd 0
ERROR_OVL_LOAD      db 'The overlay was not been loaded: '
Err1                db 'a non-existent function!',13,10,'$'
    Err2                db 'The file was not found!',13,10,'$'
Err3                db 'The route was not found!',13,10,'$'
Err4                db 'too many open files!',13,10,'$'
Err5                db 'no access!',13,10,'$'
Err8                db 'low memory!',13,10,'$'
Err10               db 'incorrect environment!',13,10,'$'
OVL1                db 'overlay1.ovl',0
OVL2                db 'overlay2.ovl',0
_DATA      ENDS

PRINT PROC  near
     push  ax
     mov   ah,09h
     int        21h
     pop   ax
     ret
PRINT ENDP
```

```
FreeMemory PROC
      mov   bx,offset DUMMY_SEGMENT
      mov   ax, es
      sub   bx, ax
      mov   cl, 4h
      shr   bx, cl
      mov   ah, 4Ah
      int   21h
      jnc   NO_ERROR
      cmp   ax, 7
      mov   dx, offset Mem_7
      je          YES_ERROR
      cmp   ax, 8
      mov   dx, offset Mem_8
      je          YES_ERROR
      cmp   ax, 9
      mov   dx, offset Mem_9
YES_ERROR:
      call  PRINT
      xor   al,al
      mov   ah,4Ch
      int   21H
NO_ERROR:
      ret
FreeMemory ENDP

FIND_PATH PROC
      push  ds
      push  dx
      mov   dx, seg DTA
      mov   ds, dx
      mov   dx, offset DTA
      mov   ah,1Ah
      int   21h
      pop   dx
      pop   ds

      push  es
      push  dx
```

```
      push  ax
      push  bx
      push  cx
      push  di
      push  si
      mov   es, KEEP_PSP
      mov   ax, es:[2Ch]
      mov   es, ax
      xor   bx, bx
COPY_CONT:
      mov   al, es:[bx]
      cmp   al, 0h
      je          STOP_COPY_CONT
      inc   bx
      jmp   COPY_CONT
STOP_COPY_CONT:
      inc   bx
      cmp   byte ptr es:[bx], 0h
      jne   COPY_CONT
      add   bx, 3h
      mov   si, offset OVL_PATH
COPY_PATH:
      mov   al, es:[bx]
      mov   [si], al
      inc   si
      cmp   al, 0h
      je          STOP_COPY_PATH
      inc   bx
      jmp   COPY_PATH
STOP_COPY_PATH:
      sub   si, 9h
      mov   di, bp
ENTRY_WAY:
      mov   ah, [di]
      mov   [si], ah
      cmp   ah, 0h
      je          STOP_ENTRY_WAY
      inc   di
      inc   si
      jmp   ENTRY_WAY
```

```
STOP_ENTRY_WAY:
     pop  si
     pop  di
     pop  cx
     pop  bx
     pop  ax
     pop  dx
     pop  es
     ret
FIND_PATH ENDP

FIND_OVL_SIZE PROC
     push ds
     push dx
     push cx
     xor  cx, cx
     mov  dx, seg OVL_PATH
     mov  ds, dx
     mov  dx, offset OVL_PATH
     mov  ah,4Eh
     int  21h
     jnc  FILE_FOUND
     cmp  ax,3
     je        Error3
     mov  dx, offset Err2
     jmp  EXIT_FILE_ERROR
Error3:
     mov  dx, offset Err3
EXIT_FILE_ERROR:
     call PRINT
     pop  cx
     pop  dx
     pop  ds
     xor  al,al
     mov  ah,4Ch
     int  21H
FILE_FOUND:
     push es
     push bx
     mov  bx, offset DTA
```

```
      mov   dx,[bx+1Ch]
      mov   ax,[bx+1Ah]
      mov   cl,4h
      shr   ax,cl
      mov   cl,12
      sal   dx, cl
      add   ax, dx
      inc   ax
      mov   bx,ax

      mov   ah,48h
      int   21h
      jnc   SUCSESS_ALLOC
      mov   dx, offset ERROR_ALLOC
      call  PRINT
      xor   al,al
      mov   ah,4Ch
      int   21h

SUCSESS_ALLOC:
      mov   BLOCK_ADDR, ax
      pop   bx
      pop   es
      pop   cx
      pop   dx
      pop   ds
      ret
FIND_OVL_SIZE ENDP

CALL_OVL PROC
      push  dx
      push  bx
      push  ax
      mov   bx, seg BLOCK_ADDR
      mov   es, bx
      mov   bx, offset BLOCK_ADDR

      mov   dx, seg OVL_PATH
      mov   ds, dx
      mov   dx, offset OVL_PATH
```

```
      push  ss
      push  sp
      mov   ax, 4B03h
      int   21h
      push  dx
      jnc   OVL_NO_ERROR

      mov   dx, offset ERROR_OVL_LOAD
      call  PRINT

      cmp   ax, 1
      mov   dx, offset Err1
      je          OVL_ERROR_PRINT
      cmp   ax, 2
      mov   dx, offset Err2
      je          OVL_ERROR_PRINT
      cmp   ax, 3
      mov   dx, offset Err3
      je          OVL_ERROR_PRINT
      cmp   ax, 4
      mov   dx, offset Err4
      je          OVL_ERROR_PRINT
      cmp   ax, 5
      mov   dx, offset Err5
      je          OVL_ERROR_PRINT
      cmp   ax, 8
      mov   dx, offset Err8
      je          OVL_ERROR_PRINT
      cmp   ax, 10
      mov   dx, offset Err10

OVL_ERROR_PRINT:
      call  PRINT
      jmp   OVL_RET

OVL_NO_ERROR:
      mov   AX,_DATA
      mov   DS,AX
      mov   ax, BLOCK_ADDR
      mov   word ptr CALL_ADDR+2, ax
```

```
      call  CALL_ADDR
      mov   ax, BLOCK_ADDR
      mov   es, ax
      mov   ax, 4900h
      int   21h
      mov   AX,_DATA
      mov   DS,AX

OVL_RET:
      pop   dx
      pop   sp
      pop   ss
      mov   es, KEEP_PSP
      pop   ax
      pop   bx
      pop   dx
      ret
CALL_OVL ENDP

MAIN  PROC  NEAR
      mov   ax,_DATA
      mov   ds,ax
      mov   KEEP_PSP, ES
      call  FreeMemory
      mov   bp, offset OVL1
      call  FIND_PATH
      call  FIND_OVL_SIZE
      call  CALL_OVL
      sub   bx,bx
      mov   bp, offset OVL2
      call  FIND_PATH
      call  FIND_OVL_SIZE
      call  CALL_OVL
      xor   al,al
      mov   ah,4Ch
      int   21h
MAIN  ENDP

_CODE       ENDS
_STACK      SEGMENT    STACK
```

```
      db    512   dup(0)
_STACK      ENDS
DUMMY_SEGMENT SEGMENT
DUMMY_SEGMENT ENDS
      END   MAIN
```