ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
Северо-Кавказский филиал
ордена Трудового Красного Знамени федерального государственного бюджетного образовательного учреждения высшего образования
"Московский технический университет связи и информатики"

Методические указания
для проведения лабораторной работы №3

по дисциплине

«СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ»

по теме

«Исследование организации резидентных программ»

Направление подготовки:
09.03.01 Информатика и вычислительная техника

Профили **Программное обеспечение и интеллектуальные системы**
Вычислительные машины, комплексы, системы и сети

Ростов-на-Дону
2019

УДК 681.3.06 (076)
ББК 32.07

Чикалов А.Н. Системное программное обеспечение. Исследование организации резидентных программ. Методические указания к лабораторным и практическим занятиям. Ростов-на-Дону: Северо-Кавказский филиал МТУСИ, 2019.- 29 с.

В пособии изложены методические рекомендации и содержательные материалы для проведения занятий по изучению механизмов межпроцессорных коммуникаций в ЭВМ. Рассматриваются модели синхронизации параллельных взаимодействующих процессов и резидентные программы, методология использования интерфейса прикладного программирования.

Пособие содержит необходимые справочные материалы.

Методические указания предназначены для студентов, обучающихся по направлению подготовки 09.03.01 Информатика и вычислительная техника, профиля Вычислительные машины, комплексы, системы и сети, Программное обеспечение и интеллектуальные системы.

Пособие предназначено для использования при изучении дисциплин Системное программное обеспечение, а также может быть использовано преподавателями и студентами при изучении родственных дисциплин и в процессе самостоятельной работы.

Учебное пособие обсуждено и одобрено на заседании кафедры ИВТ
Протокол №1 от 26.08.2019 г.

Рецензент Зав. кафедрой ИВТ д.т.н. профессор Соколов С.В.

Лабораторная работа 3. Исследование организации резидентных программ.

Часть 1. Алгоритмы синхронизации параллельных процессов

Цель

1. Выработать практические умения организации квазипараллельной работы алгоритмов при организации управления техническими системами.

2. Совершенствовать практические умения и навыки организации аппаратных и программных прерываний, оформления процедур обработки прерываний и их использования в вычислительном процессе.

3. Приобрести навыки анализа, обобщения и систематизации полученных результатов, навыки составления и оформления отчетных результатов. Привитие навыков точного и лаконичного представления докладов на вопросы технического характера.

Учебные вопросы

1. Изучение процедуры перехвата таймера.
2. Организация квазипараллельной работы потоков.

Литература для подготовки к занятию

1. Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение.- СПб.: Питер, 2001, с.221-300.
2. Олифер В.Г., Олифер Н.А. Сетевые операционные системы. – СПб.: Питер, 2001, с.140-156.
3. Финогенов К.Г. Самоучитель по системным функциям MS-DOS. - М.: Радио и связь, Энтроп, 1995, с.217-238.
4. Джордейн Р. Справочник программиста персональных компьютеров типа IBM PC, XT и AT. – М.: Финансы и статистика, 1992.
5. Данкан Р. Профессиональная работа в MS-DOS. – М.: Мир, 1993.

Содержание отчета

1. Название работы и учебные вопросы.
2. Последовательно для каждого из разделов:
   - наименование задания и разработанные программы;
   - схемы и рисунки, поясняющие механизм реализации задания;
   - пояснения к фрагментам рабочих программ;
   - ответы на вопросы, приведенные в тексте каждого конкретного задания.
3. Краткие ответы на те контрольные вопросы, которые ещё не нашли своего отражения в отчете.

Актуальность занятия

1. Потребность в синхронизации связана с совместным использованием ресурсов ВС. Она необходима для исключения тупиков и гонок при обмене

данными между потоками, разделении данных, доступе к процессору и устройствам ввода-вывода. Механизм синхронизации является одним из необходимых условий организации мультипрограммирования и основой работы любой ОС.

2. Алгоритмы синхронизации реализуются не только при работе ОС, но и при разработке программ для управления техническими объектами в рамках самостоятельных программных продуктов. Он позволяет правильно реагировать на внешние события, происходящие асинхронно и параллельно по отношению друг к другу. Это важно, т.к. подобным образом организованы все системы реального времени, в том числе и системы военного назначения.

3. Занятие имеет явно выраженную практическую направленность и связано с решением задачи автоматизации управления войсками путем использования особенностей организации современных средств вычислительной техники и его программного обеспечения.

Задачи синхронизации

Необходимость синхронизации процессов и потоков:

- нельзя заранее оценить время выполнения задачи из-за разных исходных данных, что влечет за собой разное число повторений в циклах, разные варианты ветвлений, разное время ввода-вывода;
- время поступления внешних прерываний неизвестно;
- ожидание в очереди разделяемых ресурсов неизвестно;
- могут иметь место варианты в порядке выбора потоков для выполнения.

Синхронизация – согласование скоростей путем приостановки потока до наступления некоторого события и последующей его активизации при наступлении этого события. Синхронизируют:

- ресурсы;
- данные (через буфер);
- обращение к внешним событиям.

Задачи синхронизации - исключить гонки и тупики при:

- обмене данными между потоками (забрать данные из буфера возможно только после того, как они там записаны);
- разделении данных (если они используется несколькими потоками);
- доступе к процессору;
- доступе к УВВ (находятся в состоянии ожидания до освобождения ресурса).

В зависимости от проявления названных особенностей процессы бывают:

- независимые;
- взаимодействующие.

Средства синхронизации:

- собственные средства синхронизации прикладной программы (это может быть единая (глобальная) логическая переменная о некотором событии);
- средства ОС. Это более эффективные и часто единственно возможные средства.

Форма реализации синхронизации – системные вызовы, представляемые системным и прикладным программам (для синхронизации одного процесса, разных процессов при обмене данными).

Гонки и тупики

Гонки – непредсказуемость результата из-за различных скоростей потоков, использующих разделяемые данные.

Гонки в технике - не уникальное явление. В дискретных автоматах из-за различных времен срабатывания логических элементов $t_{лэ}$ могут выполняться срабатывания элементов памяти, не предусмотренные алгоритмом.

Пример: Ведение базы данных о клиентах (рис.1.1). Каждому клиенту – отдельная запись. В записи – поля "Заявка" и "Оплата". Программа ведения базы реализована в виде отдельных потоков: потока А – внесение информации о заказах; потока В – внесение информации об оплате.

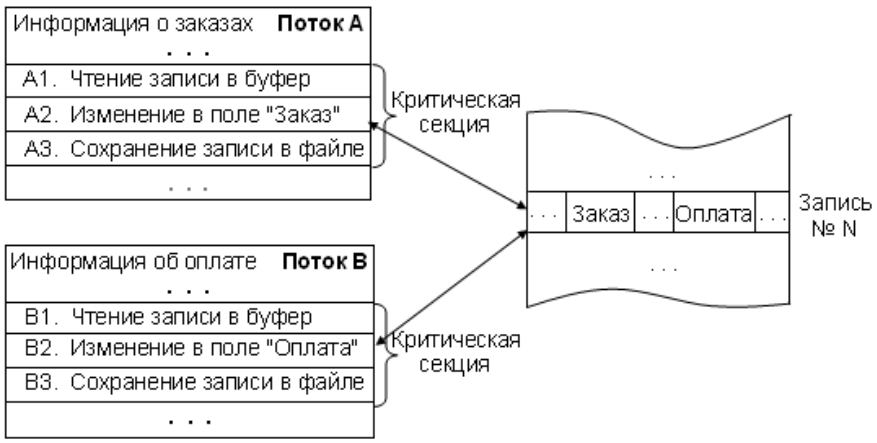

Рис.1.1. Схема работы с критическими данными

Оба потока работают с одним файлом, используя типовые алгоритмы: извлечение записи, выполнение изменений, сохранение записи на внешнем носителе. Это взаимодействующие потоки одного процесса и разделяемые ресурсы (данные). Такая модель является универсальной. Любая техническая система – это изменение состояния параметра (заказ), за этим следует новое положение исполнительных элементов (оплата), это и является резуль-

татом работы системы. Варианты итогового результата при работе таких взаимодействующих потоков представлены на рис.1.2.

Рис.1.2. Возможные результаты взаимодействующих процессов

Хранение данных осуществляется в файле, так как база данных в памяти не может находиться из-за своего большого объема (а тем более сетевая база данных). Но изменения могут происходить только в оперативной памяти.

Потоки могут прерываться по истечении кванта времени и, как следствие, могут получаться различные ситуации с состоянием данных: правильные с последовательным изменением данных на накопителе, с повреждением заявки или с повреждением информации об оплате заказа. Сложность синхронизации - в нерегулярности возникающих ситуаций. Последствия определяются:

- скоростями потоков;
- моментами прерывания процесса как следствие мультипрограммирования.

Критическая секция программы – часть программы, результат которой может быть непредсказуемым, если данные изменятся другим потоком до ее завершения.

Критические данные - при несогласованности их изменения возможны нежелательные эффекты. В примере – это "запись" из файла БД.

Способ борьбы с гонками (прием):

Взаимное исключение – работа с критическими данными только одной критической секции (одного потока). Поток может при этом находиться в ак-

тивном состоянии или приостановленном (в состоянии "готовность"). Память блокируется: пока поток не закончил работу – секция никому не дается.

Тупики – взаимная блокировка потоками вычислительного процесса (клинч – clinch, дедлоки – deadlocks). Тупик разрешается только внешними воздействиями по отношению к потоку (средствами ОС). В зависимости от особенностей потоков они могут:

- либо блокировать друг друга;
- либо образовывать очереди к разделяемым ресурсам (это нормальное явление и ситуация через некоторое время разрешится: подойдет очередь).

Условия возникновения (причины) тупиков:

- - потребность сразу в нескольких ресурсах (к сожалению это объективная потребность вычислительных процессов при мультипрограммировании);
- запрос ресурсов в разной последовательности (одинаковая последовательность исключает тупики). При одинаковой последовательности запросов на выделение ресурсов 1-й шаг уже исключает дальнейшие споры. Будет просто очередь, но не тупик.

Подходы к предупреждению тупиков:

- одинаковые последовательности запроса ресурсов на этапе программирования;
- анализ необходимых ресурсов при запуске задачи. Временно откладывается запрос при угрозе тупиков;
- ОС сама назначает правила (последовательность) выделения ресурсов (похоже на п.1, но предполагается просмотр таблицы с заявкой на ресурсы для открываемого процесса). Такой вариант более универсален и безопасен.

Подходы к распознаванию тупиков:

Признаком тупиков является тот факт, что блокированные потоки не выполняют работы, т.е. многократное выделение процессорного времени остается нереализованным, потоки все время пребывают в состоянии ожидания. При множестве блокированных потоков задача распознавания становится очень непростой.

Для распознавания тупиков используют формальные методы анализа таблицы распределения ресурсов и таблицы запросов к занятым ресурсам.

Способы обхода тупиков:

1. Снять часть потоков, освободить ресурсы для остальных;
2. Откат некоторых потоков до контрольных точек. В них запоминается вся информация для восстановления программы с данного места. Точки расставляют перед участками возможных тупиков.
3. Некоторые потоки возвращают в область подкачки.

Приемы синхронизации вычислительных процессов

1. ***Поток запрещает любые прерывания при нахождении в критической секции.***

Это выполнить просто, но опасно. По сути, управление вычислительной системой передается пользовательскому потоку. Может занять процессор надолго. А при крахе потока автоматически наступит крах системы, так как прерывания никогда не наступят и выход из неработающего потока становится невозможным.

2. Использование глобальных блокирующих переменных.

Прием применим для синхронизации потоков одного процесса. В этом случае все потоки имеют прямой доступ к памяти, не привлекая обращения к системным вызовам ОС.

Каждому набору критических данных ставится в соответствие глобальная двоичная переменная F. Если поток их занял – переменная F=0. Когда поток покидает данные – сам устанавливает переменную в 1 (т.е. данные свободны).

Если данные уже заняты (F=0), то поток переходит в режим циклического опроса, по сути - ожидания. Оно длится до истечения кванта времени, потом все повторяется при получении управления. Положительно то, что потоки могут прерываться в любом месте, в том числе в критической секции. Это одиный процесс.

Нельзя прерывать поток только между проверкой и установкой блокирующей переменной (за время приостановки может измениться занятость данных). Нарушается принцип взаимного исключения. Однако ожидание в кванте потока – это потеря процессорного времени.

Системные средства:

- в Pentium – единые неделимые команды анализа и присвоения значения логической переменной (например, BTC, BTR, BTS);
- при отсутствии команд – запрещать прерывания системными функциями на период проверки-установки.
- специальные системные вызовы для работы с критическими секциями: опрос блокирующей переменной, ее установку или выход из потока, при блокировании работы по критическим данным.

Перед изменением критических данных поток выполняет системный вызов Enter Critical Section. При этом проверяется блокирующая переменная, отражающая состояние критического ресурса. Если ресурс занят, то поток переводится в состояние ожидания и становится в очередь. При этом никакого пустого опроса не производится. Поток, который занимал этот ресурс, после выхода из критической секции выполняет системную функцию Leave Critical Section, которая устанавливает признак свободного ресурса, а ОС переводит первый в очереди поток в состояние готовности. Таким образом, исключается непроизводительная потеря времени на циклическую проверку за-

нятого ресурса. Но при минимальном ожидании становятся велики накладные расходы ОС.

3. Семафоры Дийкстры (Dijkstra)

Семафор – переменная S для синхронизации процессов, имеющая целые неотрицательные значения. Это обобщение блокирующих переменных (двоичных семафоров) для однотипных ресурсов.

Операции с семафором:

V(S) – увеличение S на 1. Выборка, наращивание и запоминание. Это этап потенциальной активизация ресурса при росте его количества;

P(S) – уменьшение S на 1, если это возможно. Если S=∅, то поток ждет пока выполнится условие S>0. Операция неделима (успешная проверка S>0 и уменьшение S). Это операция потенциального ожидания, т.к. ресурс может оказаться израсходованным.

С помощью семафоров можно организовать доступ к разделяемым ресурсам сразу нескольких потоков. Примеры наборов идентичных ресурсов:

- функционально идентичные внешние устройства (модемы, принтеры, порты);
- одинакового размера области памяти (буферы);
- одинаковые информационные структуры.

Потоки:

Задачи-писатели – пишут данные в пул.

Потоки-читатели – считывают данные из буферного пула.

При правильной работе поток-писатель – приостанавливается при занятых всех, и активизируется при освобождении хоть одного; поток-читатель – активизируется, при хоть одном занятом и приостанавливается при всех пустых буферах.

Вопрос 1.1. Изучение процедуры перехвата таймера

Сигналы прерывания таймера возникают 18,2 раз в секунду. В векторе 08h содержится адрес программы BIOS отсчета системного времени. Для того, чтобы программы пользователя могли использовать прерывания таймера, не нарушая работу системных часов, в обработчик BIOS включен вызов прерывания 1Ch, передающий управление на программу-заглушку. Перехватить прерывание таймера можно, поместив в вектор 1Ch адрес своего обработчика.

Используйте прерывания таймера для выдачи на экран (в определенное место) каждые 5 секунд количества секунд, прошедших с момента запуска программы.

```
program int1;
 uses graph,crt;
var
  old:pointer;
```

```
 t,n,x1,x2,y1,y2,gm,gd:integer;
procedure mytime1;interrupt;
 begin
  t:=t+1;
end;
begin
    asm
        mov ah,25h
        mov al,1ch
        int 21h
        mov word ptr old,bx
        mov word ptr old+2,es
        push ds
        mov ax,seg mytime1
        mov ds,ax
        mov ah,25h
        mov al,1ch
        mov dx,offset mytime1
        int 21h
        pop ds
      end;
  n:=0;
 while n<10 do
  if t>90 then
   begin
    writeln (t/18.2:8:2 );
    t:=0;  n:=n+1;
   end;
        asm
            push ds
            mov ah,25h
            mov al,1ch
            lds dx,old
            int 21h
            pop ds
          end;
     end.
```

Задание.

1. Запишите программу в конспект.
2. Составьте вербальные схемы-алгоритмов Вашего обработчика и основной программы. Определите отличия Вашего обработчика от стандартного. Зафиксируйте назначение используемых переменных.
3. Найдите и устраните неточность в приведенном тексте программы.

4. Запишите справки по новым обработчикам из рассматриваемой программы.

5. Кто инициирует прерывание 08h? Как часто выполняется прерывание 1Ch?

6. Что такое заглушка?

7. Измените программу таким образом, чтобы на экран выводились цифры с интервалом в 5 сек: 5-10-15-20 … .

Вопрос 1.2. Организация квазипараллельной работы потоков

Используйте прерывания таймера для организации квазипараллельной работы потоков. Пусть один поток рисует горизонтальную линию красного цвета, начиная с координат (1, 100), другой – линию синего цвета, начиная с координат (1,200). Для переключения между потоками используйте флаг, изменяющий свое состояние через заданные кванты времени.

Перемещение курсора в позицию с координатами (x,y) выполняется процедурой moveto(x,y). Рисование линии из текущей позиции в позицию (x,y) выполняется процедурой lineto(x,y). Установка цвета линии – процедура setcolor(<номер цвета>). Указанные процедуры доступны в подключаемом графическом модуле Graph, в котором работа осуществляется в адресуемой точечной системе координат. Поэтому имеют место отличия от текстового режима, в котором текущая позиция обозначается видимым курсором за последним отображаемым символом, а система координат задается количеством строк и столбцов для определения возможных знакомест. Отличаются также и процедуры, выполняющие одинаковые по смыслу операции (например, GotoXY и Moveto). Количество точек в графическом режиме для различных типов адаптеров может существенно различаться. Определение максимальных координат можно определить с помощью функций GetMaxX и GetMaxY.

```
program int2;
uses graph,crt;
var
        old:pointer;
        t,n,x1,x2,y1,y2,gm,gd:integer;
        flag:boolean;
procedure mytime1;interrupt;
          begin
           t:=t+1;
           if t>1 then
               begin
                  flag:= not flag;
                  t:=0;
               end;
end;
```

```
    begin
        asm
            mov ah,25h
            mov al,1ch
            int 21h
            mov word ptr old,bx
            mov word ptr old+2,es
            push ds
            mov ax,seg mytime1
            mov ds,ax
            mov ah,25h
            mov al,1ch
            mov dx,offset mytime1
            int 21h
            pop ds
        end;
      gd:=detect;
       initgraph(gd,gm,'');
       x1:=10; y1:=100;
       x2:=10; y2:=200;
      while (x2<640) and (x1<640) do
              begin
                if flag  then
                    begin
                        setfillstyle(1,4);
                        bar(x1,y1,x1+1,y1+50);
                        x1:=x1+1;
                    end
                    else
                     begin
                         setfillstyle(1,1);
                         bar(x2,y2,x2+1,y2+50);
                          x2:=x2+1;
                     end;
                    delay(20);
              end;
     readln;
    closegraph;
asm
      push ds
      mov ah,25h
      mov al,1ch
      lds dx,old
      int 21h
    pop ds
```

```
    end;
end.
```

Задание.

1. Найдите и исправьте неточность в тексте программы. Текс занесите в конспект.

2. Составьте вербальную схему-алгоритма основной программы. Определите назначение переменных.

3. Составьте вербальную схему-алгоритма обработчика.

4. Где происходит смена флагов? Каково назначение флага в задании?

5. Программа выделяет потокам одинаковые кванты времени. Как надо изменить программу, чтобы выделить потокам различные кванты времени? Проверьте решение практически.

6. Сформулируйте принципы перехвата и обработки прерывания таймера. Как оно используется для организации квазипараллельных процессов? Почему прямоугольники движутся не синхронно?

Вопросы для самоконтроля

1. Перечислите ресурсы операционной системы.
2. Каковы задачи синхронизации? Приведите определение понятия "синхронизация".
3. Что такое гонки, каковы их последствия?
4. Что такое тупик, каковы последствия их возникновения?
5. Как пользоваться сигнальной переменной?
6. Каким образом осуществляется перехват прерывания таймера? Каковы номера используемых прерываний?
7. Что такое блокирующая переменная? Поясните механизм ее использования.
8. Что выполняют функции 25H и 35H прерывания 21H? Какие и где требуются данные при вызове и какие данные возвращаются?
9. Как передаются в качестве параметров области оперативной памяти при вызове системных функций?
10. Приведите классификацию прерываний.
11. В чем различие между программными и аппаратными прерываниями?
12. Поясните, в чем заключается параллельность процессов для рассматриваемых заданий.
13. Для рассматриваемых заданий укажите моменты времени, в которые проявляется работа выявленных вами участков программы.

Часть 2. Резидентные программы

Цель

1. Выработать практические умения разработки и использования резидентных программ.

2. Закрепить навыки эксплуатации ПЭВМ в ОС MS DOS и использования интерфейсной оболочки NC (VC).

4. Приобрести навыки анализа, обобщения и систематизации полученных результатов, навыки составления и оформления отчетных результатов. Привитие навыков точного и лаконичного представления докладов на вопросы преподавателя.

Учебные вопросы

1. Исследование структуры и механизма работы резидентной программы, активизирующейся по нажатию клавиши.

2. Исследование процесса активизации резидентной программы по сигналам таймера.

Литература для подготовки к занятию

1. Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение. Питер. 2001, с.37-44.
2. Олифер В.Г., Олифер Н.А. Сетевые операционные системы. Питер. 2001.
3. Финогенов К.Г. Самоучитель по системным функциям MS-DOS. М. Радио и связь. Энтроп. 1995. с.238-246, 257-271.
4. Скэнлон Л. Персональные ЭВМ IBM PC и XT. Программирование на языке ассемблера. М.: Радио и связь, 1989.
5. Нортон П. Персональный компьютер фирмы IBM и операционная система MS-DOS. М.: Радио и связь, 1991.
6. Скляров В.А. Применение ПЭВМ. Кн.1. Операционные системы ПЭВМ. Практическое пособие. - М.: ВШ, 1992.

Содержание отчета

1. Название лабораторной работы и учебные вопросы.
2. Последовательно для каждого из разделов исследования:
   - задания и разработанные программы;
   - схемы и рисунки, поясняющие механизм перехвата аппаратных прерываний от таймера;
   - пояснения к фрагментам рабочих программ.
3. Выводы о целесообразности реализации аппаратных прерываний с использованием нестандартных процедур обработки прерываний.
4. Краткие ответы на те контрольные вопросы, которые ещё не нашли своего отражения в отчете.

Вопросы для самопроверки

1. Каким образом осуществляется обращение к функциям DOS и BIOS?
2. Поясните понятие резидентной программы и приведите примеры.
3. Поясните состав структуры резидентной программы, назначение элементов .
4. Поясните порядок активизации резидентной программы.
5. Каким образом осуществляется обращение к резидентной программе?
6. Поясните механизм процедуры прерывания, проиллюстрируйте ее.
7. В чем различие между программными и аппаратными прерываниями?
8. Поясните назначение вектора прерываний, способ его использования.
9. Для чего в составе машин IBM PC имеется таймер?
10. Поясните прикладное значение обработки прерываний от таймера.
11. Поясните организацию аппаратных прерываний в машинах IBM PC.

Актуальность занятия

1. Механизм прерываний является одним из способов организации мультипрограммирования и основой работы любой ОС. Он позволяет реагировать на внешние события, происходящие асинхронно вычислительному процессу. Это важно, т.к. подобным образом организованы все системы реального времени, в том числе и системы военного назначения
2. Это эффективный способ вызова процедур ОС, в том числе резидентных, использования разработчиком имеющихся ресурсов, генерации нестандартных процедур обработки.
4. Занятие имеет явно выраженную практическую направленность и связано с решением задачи автоматизации управления войсками путем использования особенностей организации современных средств вычислительной техники и его программного обеспечения.
5. Это инструмент подключения драйверов для новых устройств.

Организация адресного пространства оперативной памяти

Оперативная память компьютера **физически** представляет собой упорядоченную последовательность ячеек размером по 1 байту. Каждая ячейка имеет адрес. Диапазон адресов, начинаясь с нуля, заканчивается цифрами, характеризующими предельный размер имеющейся фактической физической памяти. В этот диапазон входят как энергозависимая память, так и постоянная память, реализованная в виде микросхем на материнской плате.

Логически память формируется из сегментов. **Сегмент** – это любая неразрывная область памяти размером до 64К. Адрес начала сегмента кратен

16 или одному **параграф**у. Параграфы нумеруются от 0 до 65535 и начинаются с адреса 0:

№ параграфа	Адрес параграфа
0	0h
1	10h
2	20h
3	30h
. . .	. . .
65535	FFFF0h

Адрес начала сегмента соответствует номеру первого параграфа, с которого начинается сегмент. Этот адрес называется **сегментным** и занимает 2 байта. Для адресации ячейки внутри сегмента требуется еще 2 байта чтобы указать любой адрес внутри максимальной области из 64К. Этот адрес называется **смещением**.

Для MS DOS адрес ячейки формируется по формуле:

Адрес ячейки (байта) = сегмент * 16 + смещение.

Умножение на 16 соответствует сдвигу на 4 разряда влево. Поэтому весь адрес формируется из 20 адресных линий и способен адресовать память размером 1 Мб. Например, сегмент равен 312Fh, а смещение – 25Ah . После умножения на 4 получаем адрес 312F0h. Суммирование со смещение дает адрес ячейки – 3154Ah.

При адресации команд сегментный адрес хранится в регистре CS, смещение - в IP. Их совокупность образует узел, который для упрощения называют счетчиком команд.

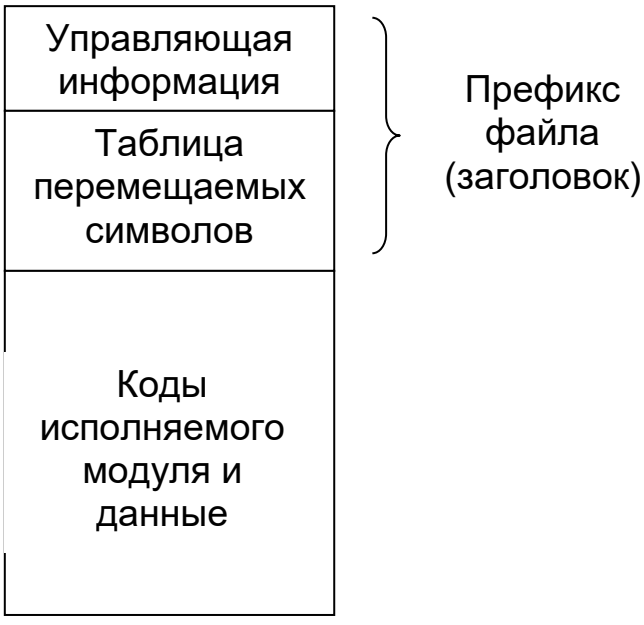

Рис.2.1. Структура .exe файла

Операционная система ищет свободную память, перебирая параграфы. Таким именно образом она автоматически назначает сегменты.

Структура исполняемого .EXE-файла

На диске исполняемый файл (с расширением .exe) состоит из заголовка (префикса) и кодов исполняемого модуля и данных (рис.2.1). Префикс обозначают аббревиатурой PSP - Program Segment Prefix –префикс сегмента

программы.

Структура управляющей информации представлена в табл.1. Таблица перемещаемых символов содержит элементы, каждый из которых представляет собой смещение адресной константы, которую необходимо настроить относительно адреса загрузки, т.е. стартового сегмента. В исполняемом модуле адресные константы необходимо настроить для конкретного адреса загрузки. В файле они вычислены относительно начала сегмента (это относительный адрес). При выполнении они должны иметь абсолютные адреса.

Таблица 1

Управляющая информация в заголовке (префиксе) файла

Поле	Смещение	Содержание
1	00H	4DH, 5AH - идентификатор EXE файла
2	02H	Длина файла по модулю 512 (остаток от деления длины файла на 512)
3	04H	Длина файла в блоках по 512 байтов
4	06H	Количество элементов в **таблице перемещаемых символов**
5	08H	Размер префикса в параграфах (по 16 байтов)
6	0AH	Минимальное количество параграфов, необходимых после загруженной программы (обычно 0)
7	0CH	Максимальное количество параграфов, необходимых после загруженной программы
8	0EH	Смещение SS в выполняемом модуле (в параграфах)
9	10H	Значение SP при получении управления выполняемым модулем
10	12H	Контрольная сумма (отрицательная сумма всех слов в файле с игнорированием переполнения)
11	14H	Значение IP, когда выполняемый модуль получает управление
12	16H	Смещение CS в выполняемом модуле (в параграфах)
13	18H	Начало таблицы перемещаемых символов (смещение с начала файла в байтах)
14	1AH	Номер перекрытия

Этапы загрузки и выполнения программы

I. Загрузка:

1) Управляющая информация заголовка считывается в рабочую область DOS;
2) Определяется размер выполняемого модуля (по полям 2, 3, 5, 6);
3) Находится 1-й свободный блок памяти (сегмент), размер которого достаточен для PSP и исполняемого модуля;
4) Строится PSP для программы. Сегмент после PSP назначается стартовым сегментом. Структура PSP программы показана в табл.2 (не путайте с PSP файла);

Таблица 2

Содержание PSP программы (Program Segment Prefix)

Поле	Смещение		Длина	Содержание
	(10)	(16)		
1	0	0	2	Инструкция INT 20H (завершение программы)
2	2	2	2	Размер памяти в блоках по 16 байтов (параграфах)
3	4	4	1	Резервировано
4	5	5	5	Дальний переход (CALL сегмент: смещение) Второй байт (смещение) содержит количество свободных байтов в сегменте
5	10	A	4	Адрес завершения (вектор 22 H)
6	14	E	4	Адрес обработки Ctrl-Break (вектор 23H)
7	18	12	4	Адрес обработки ошибок (вектор 24H)
8	22	16	22	Используется оперативной системой ДОС-16
9	44	2C	2	Адрес окружения (только сегмент, смещение - 0)
10	46	2E	34	Рабочая область ДОС
11	80	50	3	Инструкции INT 21H и RET FAR
12	83	53	2	Резервировано
13	85	55	7	Префикс первого FCB (File Control Block)
14	92	5C	9	Первый FCB
15	101	65	7	Префикс первого FCB
16	108	6C	9	Второй FCB
17	128	80	1	Длина параметров командной строки
18	129	81	127	Параметры командной строки
19	128	80	128	DTA (Disk Transfer Area) - по умолчанию

5) Исполняемый модуль загружается с начала стартового сегмента;
6) Таблица перемещаемых символов считывается в рабочую область DOS. Пересчитываются адресно-зависимые параметры программы.

В качестве примера в таблице представлен фрагмент загружаемой программы в мнемокодах с необходимыми параметрами трансляции. Адреса трансляции указаны в предположении, что программа размещается с начального адреса сегмента, т.е. смещение равно 0. При загрузке свободное место в ОП найдено с 1-го параграфа (абсолютный адрес равен 16). Команды пере-

сылки занимают по 1 байту, команда безусловного перехода – 3 байта, команда пересылки в память – 3 байта.

Таблица перемещаемых символов и фрагмент программы

Таблица перемещаемых символов	Метки	Адрес при трансляции в дес. системе счисления	Мнемокод команды	Адрес фактической загрузки, в дес. системе счисления
101		0	MOV AH, BH	16
104	M1:	1	MOV DL, CH	17
			. . . . .	
		100	JMP M1 (1+16=17)	116
		103	MOV AX, VAR1 (200+16=216)	119
			. . . . .	
	VAR1	200	DW 157 (слово)	216

В таблице перемещаемых символов имеется две записи, указывающие на адреса в загружаемой программе, которые должны быть пересчитаны при загрузке: адрес смещения 101, с которого начинается в команде JMP адрес безусловного перехода, и адрес 104, с которого начинается адрес памяти в команде пересылки.

Алгоритм пересчета следующий: читается код адреса по первому коду (101) таблицы перемещаемых символов. В данном примере это адрес метки M1, имеющий при трансляции численное значение 1. К нему добавляется фактическое смещение (16) и записывается опять по тому же адресу (101). Эти расчеты показаны в таблице в скобках после команды JMP. После такой настройки адресов обращение к метке M1 (фактический новый адрес в ОП - 17) будет осуществляться правильно. Аналогично выполняется пересчет адреса по коду 104. Выполняется пересчет системной программой загрузки – перемещаемым загрузчиком.

7. Назначаются регистры:
   - Данных ES и DS;
   - стека SS : SP;
   - команд CS : IP.
   -

II. Выполнение

Управление передается CS:IP (счетчику команд). С этого момента начинают выполняться команды загруженной программы.

III. Завершение выполнения программ. Программа может завершиться выполнением следующих обработчиков:

- INT 20h – завершить процесс;
- INT 21h с функцией 00h – завершить процесс;
- INT 21h с функцией 4Ch – завершить процесс с кодом возврата.

В самом начале PSP (смещение 0) записана команда INT 20h, поэтому завершение может произойти и переходом к началу PSP. Выполняются во всех случаях следующие операции:

- освобождает место в ОП (все блоки программы);
- восстанавливаются в таблице векторов вектора из PSP прерванного процесса: 22h – адрес перехода при завершении, 23h – обработка клавиш CTRL / C, 24h – обработка критических ошибок;
- управление передается по адресу завершения (прерывание INT 22h – адрес перехода при завершении процесса). Фактически управление передается так называемому родительскому процессу, которым чаще всего является сама ОС.

Особенности исполняемых .COM – файлов

COM- файлы не имеют префикса (заголовка). Они строятся так, чтобы не содержали адресных констант, зависимых от адреса загрузки программы в память. Все в программе имеет смещение от начала сегмента, включая данные и стек программы. По этой причины программа имеет следующие особенности:

1) Размер программы не более 64К (это максимальный размер сегмента);
2) ОС ищет сразу большой размер свободной оперативной памяти, строит PSP программы и сразу за ним размещает коды программы и все остальное непосредственно из файла;
3) Сегментным регистрам CS, DS, SS, ES присваивается значение сегмента программы (это сегментный адрес PSP). IP получает адрес 100h , т. к. размер PSP 256 байт. По этой причине трансляция программы осуществляется с применением оператора ORG 100h. Стек размещается в конце адресного пространства из 64К, поэтому SP=FFFEh.

Основы организационных резидентных программ

Большинство программ загружаются, выполняются и завершаются. Их называют **транзитными**, потому что они находятся в оперативной памяти

только во время выполнения (транзитом, "проездом") Это стандарт языков высокого уровня: используют только стандартные обслуживающие функции DOS.

При использовании всех обслуживающих функций (на ассемблере это допустимо) возможности шире. Возможно, например:

- создавать резидентные программы, получающие управление при определенных условиях;
- оверлейные структуры – программа из частей, которые загружаются последовательно на одно и то же место и запускаются на выполнение;
- из программы выполнить другую программу или внутреннюю команду DOS;
- программа может установить код завершения. Этот код доступен родительскому процессу при помощи специальной системной функции (если родительский процесс является программа COMMAND.com, код завершения можно проверить средствами командных файлов);
- память можно распределять динамично.

Резидентные программы находятся постоянно в ОП, никак себя не проявляют и реагируют (запускаются) на запросы пользователя или события в системе. Таким образом, они обеспечивают ее работу. Программы часто называют TSR - Terminate and Stay Resident - выполниться и остаться в памяти. Они являются программами исполняемыми, но чаще .com-типа по причине своей компактности. Такими программами являются:

- драйверы устройств;
- интерфейсные оболочки DOS;
- средства шифрации и защиты данных;
- русификаторы;
- обслуживающие программы типа электронных блокнотов, калькуляторов и др.

На вычислительный процесс резидентные программы не влияют (до вызова), но занимает память. Их может быть в ОП несколько.

Этапы работы с резидентной программой:

1) Загрузка программы в ОП целиком;
2) Передача управления инициализирующей части, которая
   - устанавливает векторы прерывания;
   - настраивает программу по необходимым параметрам;
3) Завершение работы с оставлением в памяти резидентной части. Инициализирующая часть при этом отбрасывает в целях высвобождения дополнительной оперативной памяти.

Способы запуска резидентной программы

При запуске резидентной программы необходимо передать управление и передать параметры. Различают следующие способы:

1. Вызвать как подпрограмму (оператором CALL). Это требует текущей активной программы взаимодействующая с родительской, наличия межпрограммных соглашений по взаимодействию (межпрограммный интерфейс). Но вместе с тем, выделив общие для группы транзитных программ функции и оформив их резидентными, можно упростить структуру и объем транзитных программ, такого многопрограммного комплекса;
2. Использовать механизм аппаратных прерываний (асинхронный способ). Это применяется для таймера, клавиатуры, портов и других внешних устройств, широко используется системными и прикладными программами для принтеров, программ календарей и часов, русификаторов, калькуляторов, электронных блокнотов, резидентных электронных справочников, БД. Адрес помещается в имеющийся вектор прерываний и организуется взаимодействие с системным обработчиком;
3. Программным прерыванием (синхронный способ). Это основа взаимодействия с функциями DOS и BIOS, иногда используется прикладными резидентными программами, в частности, при отладке программ (INT 3). Весьма изящный метод – использование свободных векторов пользователя G0 ÷66h. В процессе инициализации в них размещается адрес резидентной программы. Для активизации резидентной программы используется команда INT 60h в транзитной программе;
4. Специальное мультиплексное прерывание. Прерывание INT 2Fh обеспечивает взаимодействие с другими процессами и расширениями MS DOS.

Структура резидентной программы

Резидентные программы чаще пишутся в формате Com. Резидентные программы обычно состоят из двух частей: инициализирующей и рабочей (резидентной). В тексте программы резидентная часть располагается вначале, инициализирующая - за ней. При первом вызове программы она загружается в память целиком и управление передается секции инициализации, которая заполняет или модифицирует векторы прерываний и с помощью функции 31h прерывания 21h завершает программу, оставляя в памяти ее резидентную часть. Эта функция, закрепив за резидентной программой необходимую память, передает управление командному процессору, и вычислительная система переходит в исходное состояние. Для того, чтобы активизировать резидентную программу, ей нужно передать параметры и управление.

Процесс выполнения выглядит следующим образом:

1) При первом запуске управление передается на начало процедуры;
2) По команде JMP, которая стоит самой первой, осуществляется переход на секцию инициализация, и готовятся условия для активизации в резидентном состоянии. В частности, могут устанавливаться

необходимые вектора прерывания для последующего вызова резидентной программы;

3) Инициализация заканчивается выполнением системного вызова по INT 21h с функцией 31h – завершение программы с оставлением указанной части в ОП. Иногда функцию формулируют - завершить и оставить резидентной. Входными параметрами являются AH = 31h; AL – код возврата, с которым предлагается завершить программу (0 - успешное выполнение, ≠0 – ошибка), DX – объем резервируемой памяти (в параграфах). Выходных параграфов функция не имеет.

Функция 31h выполняет:

- резервирует память в объеме заявленных параграфов, начиная от PSP (при обычном завершении освобождается все пространство);
- сбрасывает файловые буферы. Дескрипторы файлов или устройств, открытых самим процессом, закрывает;
- восстанавливает из PSP векторы 22h – адрес перехода при завершении; 23h – обработка CTRL/C; 24h – обработка критических ошибок;
- передает управления родительскому процессу по INT 22h.

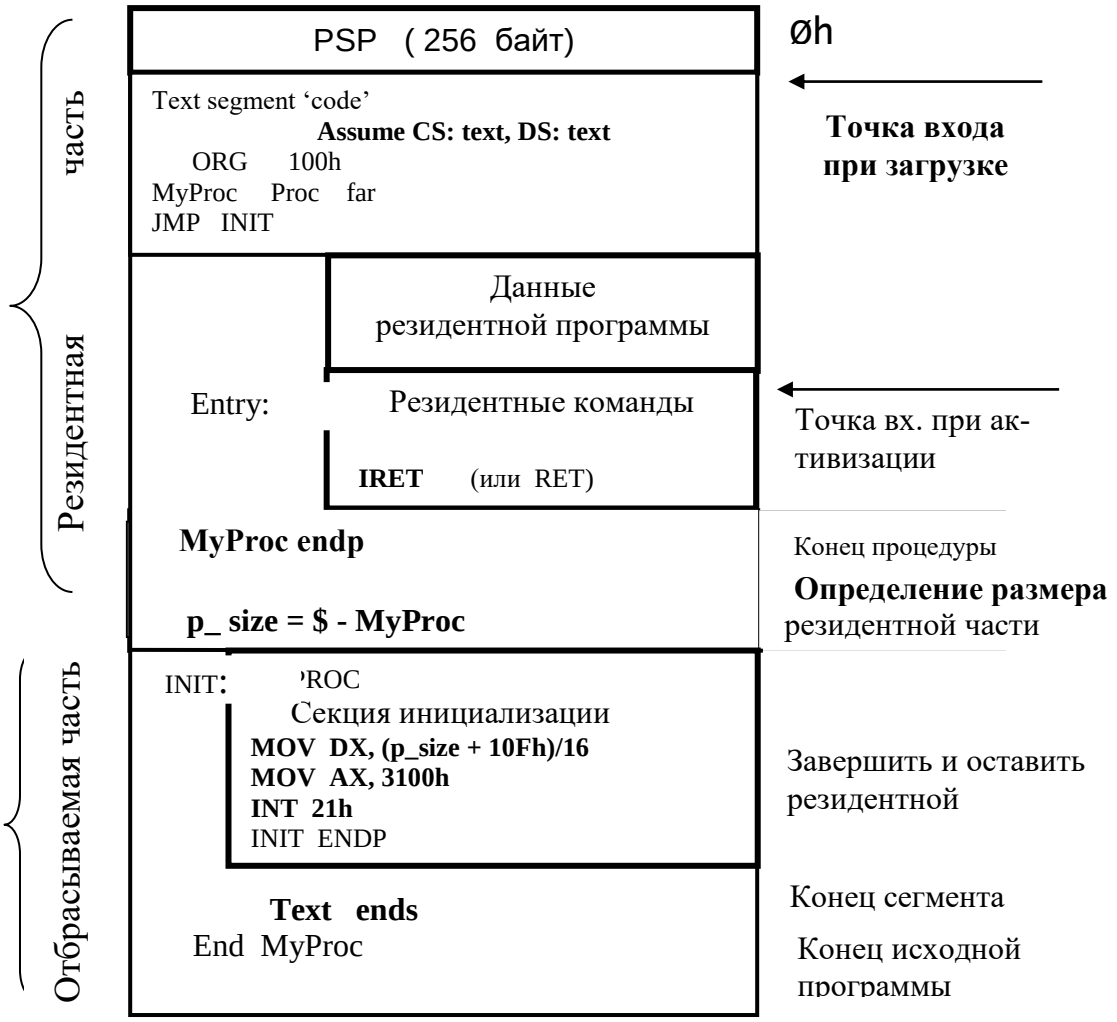

Рис.2.2. Структура резидентной программы

Вопрос 2.1. Исследование структуры и механизма работы резидентной программы

Введите, оттранслируйте и запустите приведенную резидентную программу, изменяющую цвет границы экрана на фиолетовый при нажатии на «серый плюс» и возвращающую черный цвет границы при нажатии на «серый минус».

Работа осуществляется в "чистом ассемблере". Поэтому необходимо следовать определенным этапам:

1. Подготовить исходный текст программы в простейшем редакторе с расширением .asm. Это рационально сделать с помощью встроенного редактора интерфейсной оболочки VC.com, реализуя команды ПРАВКА, ЗАПИСЬ, ВЫХОД, УДАЛЕНИЕ и т.д. Например, имя файла получится ZADAN1.asm;
2. Выполнить трансляцию для получения перемещаемого объектного модуля исходной программы (расширение .obj). Для этого необходимо в командной строке разместить имя транслятора TASM.exe (выбрать его в окне VC-оболочки и нажать Ctrl+Enter) и имя подготовленного исходного текста ZADAN1.asm (аналогично, выбрать имя файла и нажать Ctrl+Enter). В результате всех операций в командной строке должно сформироваться

TASM.exe ZADAN1.asm

Если все правильно – нажмите ENTER для начала трансляции. После окончания должен появиться файл объектного модуля ZADAN1.obj;

3. Выполните компоновку загрузочного модуля с помощью компоновщика. Для этого сформируйте по тем же правилам в командной строке следующее выражение

LINK.exe ZADAN1.obj /t /x

Параметр /t – предписывает создать файл типа .com.

/x – запрещает создавать вспомогательный файл с расширением .map. После компоновки должен получиться файл ZADAN1.com;

4. Запустите файл на выполнение. Попытайтесь это сделать из операционной оболочки, выбрав файл и нажав ENTER. Второй вариант возможен при создании ярлыка с расширенным окном.

Программа использует перехват двух прерываний: 09h и 28h. Перехват прерывания 09h выполняется резидентной программой для получения сигналов активации (нажатие на «серый плюс» или «серый минус»). Перехват прерывания 28h объясняется следующим. Программы обработки прерываний являются нереентерабельными, то есть не допускающими вызов других прерываний внутри себя. Однако резидентная программа, активизируе-

мая прерыванием от клавиатуры, должна для изменения цвета края экрана вызвать прерывание 10h. Для преодоления нереентерабельности в DOS включено специальное прерывание 28h, вызываемое функциями ввода с клавиатуры. Системный обработчик этого прерывания содержит единственную команду возврата из процедуры обработки и поэтому никак не нарушает ход программы. Такой обработчик называют заглушкой. Однако пользователь может поместить в этот вектор адрес своей программы. Поскольку это прерывание возникает только при выполнении функций, использующих стек ввода-вывода, в обработчике этого прерывания могут использоваться вызовы системных функций.

Текст программы.

```
text segment 'code'
    assume CS:text,DS:text
    org 100h
int1 proc far
     jmp init
old_28 dd 0; переменная для хранения адреса старого обработчика 28h
old_9   dd 0; переменная для хранения адреса старого обработчика 09h
fl     db 0; флаг, фиксирующий активацию резидентной программы
new_9  proc ;новый обработчик прерываний 09h
    push ax
    in al,60h
    cmp al,78; серый плюс?
    je plus
    cmp al,74; серый минус?
    je minus
    pop ax
    jmp cs:old_9; не нам, передаем управление старому обработчику
plus : mov cs:fl,1; устанавливаем флаг в 1, если «серый плюс»
     jmp ee
minus: mov cs:fl,2; устанавливаем флаг в 2, если «серый минус»
ee:   pop ax
     jmp cs:old_9; передаем управление старому обработчику
new_9 endp
new_28 proc; новый обработчик прерываний 28h
     cmp cs:fl,0
     jg go; если флаг больше нуля, значит программу активизировали
     jmp cs:old_28; иначе передаем управление старому обработчику
go:  push ax
    push bx
    cmp cs:fl,1
    jne bla
```

```
        mov bh,0dh; если флаг равен единице, зашлем в регистр bh фиолетовый
        jmp color
    bla:mov bh,0; если флаг равен двум, зашлем в регистр bh черный
    color: mov ax,1001h
        int 10h; сменим цвет границы
        pop bx
        pop ax
        iret
    new_28 endp
    size_p=$-int1; определим размер резидентной части
    init proc; секция инициализации
        mov ax,3509h
        int 21h
        mov word ptr old_9,bx
        mov word ptr old_9+2,es; запомнили адрес старого обработчика 09h
        mov ax,3528h
        int 21h
        mov word ptr old_28,bx
        mov word ptr old_28+2,es; запомнили адрес старого обработчика 28h
        mov ax,2509h
        mov dx,offset new_9
        int 21h; поместили адрес нашего обработчика в 09h
        mov ax,2528h
        mov dx, offset new_28
        int 21h ; поместили адрес нашего обработчика в28h
        mov dx,(size_p+10fh)/16; передаем размер резидентной части в параграфах
        mov ax,3100h
        int 21h; завершаем программу и оставляем в ОП ее резидентную часть
    init endp
    text ends
        end int1
```

Задание.

1. Занесите текст программы в конспект.
2. На основе анализа текста программы, составьте вербальные алгоритмы обработчиков и программы. Определите назначение переменных.
3. Модифицируйте программу так, чтобы по нажатию «серого плюса» устанавливался красный цвет края экрана, а по нажатию «серого минуса» - зеленый.

4. Зафиксируйте справочные данные для используемых новых обработчиков (функция 31h INT 21h, INT 28h).

5. Докажите, что программа является резидентной.

Вопрос 2.2. . Исследование процесса активизации резидентной программы по сигналам таймера

Следующая программа выводит на экран сообщение "We are here" через определенные промежутки времени. В связи с невозможностью использовать прерывания сообщение выводится побайтно прямо в видеобуфер, находящийся по адресу 0b800h. Каждый символ видеобуфера занимает 2 байта. Первый байт - это сам символ, второй - его атрибуты.

Формат атрибута:

7	4-6	3	0-2
Мерцание	Цвет фона	Яркость	Цвет букв

Введите, оттранслируйте и запустите программу.

```
text segment 'code'
    assume CS:text,DS:text
    org 100h
int1 proc far
     jmp init
sm     dw 1 ; смещение относительно начала видеобуфера
t      db 0; переменная для подсчета времени
p db 'We are here'; строка сообщения
size_t=$-p
new_t  proc; обработчик прерываний таймера
    inc cs:t; увеличим t на 1
    cmp cs:t,50; t=50?
    jne e; если нет, ничего не делаем, идем на конец процедуры
    add cs:sm,size_t; иначе сместимся в текстовом видеобуфере на длину сообщения
    push ax; сохраним в стеке все используемые регистры
    push cx
    push es
    push bx
    push si
    mov ax,0b800h; настроим es на начало видеобуфера
    mov es,ax
    mov bx,cs:sm; bx = смещение
    mov si,0; установим указатель на начало строки сообщения
    mov cx,size_t;занесем в cx длину сообщения
```

```
 l: mov al,cs:p[si]; выведем очередной символ строки в видеобуфер
    mov es:[bx],al
    inc bx; сместимся в следующую ячейку видеобуфера
    mov al,84h; запишем в нее  атрибут символа- красный, мерцающий
    mov es:[bx],al
    inc si; сместимся к следующему символу строки
    inc bx; в следующую ячейку видеобуфера
    loop l; cx=cx-1, и если cx<>0 вернемся в начало цикла
    add cs:sm,size_t;увеличим смещение на длину сообщения
    pop si;восстановим все регистры
    pop bx
    pop es
    pop cx
    pop ax
  e: iret
new_t endp
size_p=$-int1;длина резидентной части
init proc;секция инициализации
    mov ax,251ch; помещаем наш обработчик в вектор 1ch
    mov dx,offset new_t
    int 21h
    mov dx,(size_p+10fh)/16
    mov ax,3100h; завершаем программу, оставляя ее резидентной
    int 21h
init endp
text ends
    end int1
```

Задание

1. Составьте вербальный алгоритм программы.
2. Измените текст выводимого сообщения.
3. Попробуйте изменить длительность промежутков между сообщениями а) в меньшую сторону, б) в большую сторону.
4. Измените атрибуты символов выводимой строки.
5. Зафиксируйте справочные данные для новых обработчиков.

Вопросы для самоконтроля

1. Каким образом осуществляется обращение к функциям DOS и BIOS?
2. Поясните понятие резидентной программы и приведите примеры.
3. Поясните состав структуры резидентной программы, назначение элементов.
4. Поясните порядок активизации резидентной программы.

5. Каким образом осуществляется обращение к резидентной программе?
6. Поясните назначение вектора прерываний, способ его использования.
7. Каковы механизмы защиты резидентной программы от повторной установки? Почему это опасно?
8. Какие существуют способы выгрузки резидентной программы?
9. Почему инициализирующая часть программы располагается в конце исходного текста ?
10. Назовите основные этапы создания исполняемого файла?
11. Как определить, что программа является резидентной?