Системное программирование

Лекция №15

Модульное программирование

Структурное программирование – методология программирования, базирующаяся на системном подходе к анализу, проектированию и реализации программного обеспечения:

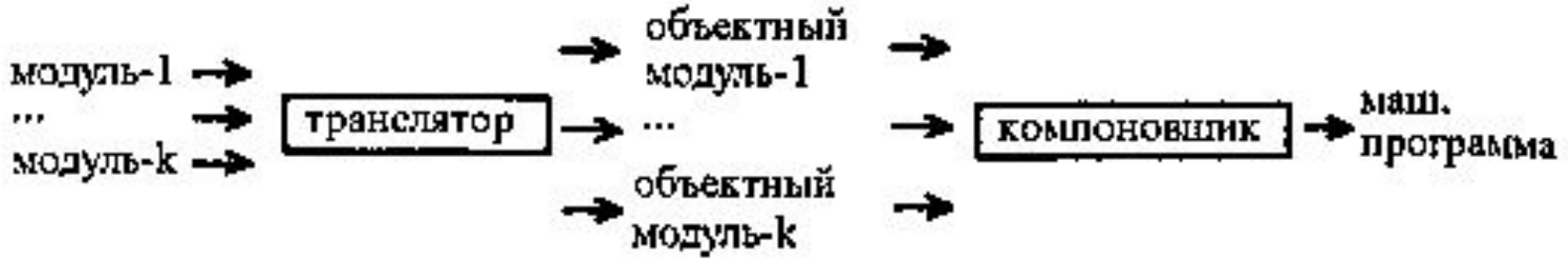
- Сложная задача разбивается на более мелкие, функционально лучше управляемые задачи. Каждая задача имеет один вход и один выход.
- Логически задача должна состоять из минимальной, функционально полной совокупности достаточно простых управляющих структур.
- Разработка программы должна вестись поэтапно. На каждом должно решаться ограниченное число четко поставленных задач с ясным пониманием их значения и роли в контексте всей задачи.

Модульное программирование:

Функциональная декомпозиция задачи – разбиение задачи на ряд более мелких, функционально самостоятельных подзадач – *модулей*. Модули связаны между собой только по входным и выходным данным.

Первый вариант объединения модулей

Второй вариант объединения модулей



Формы организации управляющих связей:

- Использование механизма макроподстановок.
- Использование механизма процедур, написанных на ассемблере.
- Использование механизма подпрограмм, написанных на разных языках программирования и соединяемых в единый модуль на этапе компоновки.
- Использование механизма динамического вызова исполняемых модулей и подключения библиотек .dll для операционной системы Windows.

Формы организации информационных связей:

- Использование общих областей памяти и общих программно-аппаратных ресурсов микропроцессора.
- Унифицированная передача аргументов при вызове модуля.
- Унифицированная передача аргументов при возврате управления из модуля.

Модульное программирование

Директива EXTRN предназначена для объявления некоторого имени внешним по отношению к данному модулю.

Директива PUBLIC предназначена для объявления некоторого имени, определенного в этом модуле и видимого в других модулях.

EXTRN имя:тип,…, имя:тип

PUBLIC имя,…,имя

имя – идентификатор, определенный в другом модуле. В качестве идентификатора могут выступать:

- имена переменных, определенных директивами типа DB,DW и т.д.;
- имена процедур;
- имена констант, определенных операторами = и EQU.

если *имя* – это имя переменной, то *тип* может принимать значения BYTE, WORD, DWORD; если *имя* – это имя процедуры, то *тип* может принимать значения NEAR или FAR; если *имя* – это имя константы, то *тип* должен быть ABS.

Модульное программирование

```
;Модуль 1
MASM
.MODEL  SMALL
.STACK  256
.DATA
…
.CODE
proc_1    PROC
…
proc_1    ENDP
proc_2    PROC
…
proc_2    ENDP
;объявляем процедуру proc_1 видимой извне
PUBLIC  proc_1
start:
    MOV  AX,@data
…
ENDstart
```

Модульное программирование

```
;Модуль 2
MASM
.MODEL  SMALL
.STACK  256
.DATA
…
.CODE
EXTRN   proc_1    ;объявляем процедуру proc_1 внешней
start:
    MOV  AX,@data
…
    CALL proc_1    ;вызов proc_1 из модуля 2
ENDstart
```

Трансляция:

TASM.EXE PR1 результат — PR1.OBJ

TASM.EXE PR2 результат — PR2.OBJ

Компоновка:

TLINK PR2+PR1 результат — PR2.EXE
(Главный модуль должен быть первым!)

Библиотека:

TLIB LIB1.LIB + PR1.OBJ результат — библиотечный файл LIB1.LIB, содержащий модуль 1.

Модульное программирование

Аргумент – это ссылка на некоторые данные, которые требуются для выполнения возложенных на модуль функций и размещенных вне модуля. Рассматривают понятия *формального* и *фактического* аргументов.

Переменная – это нечто, размещенное в регистре или ячейке памяти, что может в дальнейшем подвергаться изменению.

Константа – данные, значение которых никогда не изменяется.

Варианты передачи аргументов в модуль (процедуру):

- через регистры;
- через общую область памяти;
- через стек;
- с помощью директив EXTRN и PUBLIC

Передача аргументов через регистры:

- небольшое количество регистров, доступных для пользователя;
- необходимость постоянно контролировать содержимое регистров;
- размер передаваемых данных — только 8, 16 или 32 бита

(в других случаях нужно передавать не сами данные, а указатели на них)

Метод широко применяется при вызове функций DOS

Передача аргументов через общую область памяти

Недостатком этого способа в реальном режиме работы МП является отсутствие средств защиты данных от разрушений, так как нельзя проконтролировать соблюдение правил доступа к этим данным.

Модульное программирование

```
;prg1.asm
include   mac.inc  ;подключение файла с макросами
stk   segment stack
      db   256 dup (0)
stk   ends
common_data      segment para common 'data'     ;начало общей области
памяти
buf   db   15 DUP (' ')   ;буфер для хранения строки
temp      dw   0
common_data      ends
      extrn      PutChar:far,PutCharEnd:far
code      segment ;начало сегмента кода
      assume  cs:code,es:common_data
main      proc
      mov ax,common_data
      mov es,ax
;вызов внешних процедур
      call  PutChar
      call  PutCharEnd
      push      es
      pop ds
      _OutStr  buf
exit:
      _Exit      ;стандартный выход
main      endp      ;конец главной процедуры
code      ends
      end main
```

Модульное программирование

```
;prg2.asm
include   mac.inc  ;подключение файла с макросами
stk   segment stack
      db   256 dup (0)
stk   ends
pdata     segment para public 'data'
mes db    'Общий сегмент',0ah,0dh,'$'
temp1     db   ?
temp2     dd   ?
temp3     dq   ?
pdata     ends
      public     PutChar,PutCharEnd
common_data segment para common 'data'  ;начало общей
                  ;области памяти
buffer    db   15 DUP (' ')   ;буфер для формирования
строки
tmpSI     dw   0
common_data      ends
code      segment ;начало сегмента кода
      assume  cs:code,es:common_data,ds:pdata
PutChar procfar   ;обьявление процедуры
      cld
```

```
        mov si,0
        mov buffer[si],'Р'
        inc  si
        mov buffer[si],'а'
        inc  si
        mov buffer[si],'б'
        inc  si
        mov buffer[si],'о'
        inc  si
        mov buffer[si],'т'
        inc  si
        mov buffer[si],'а'
        inc  si
        mov buffer[si],'е'
        inc  si
        mov buffer[si],'т'
        inc  si
        mov buffer[si],'!'
        inc  si
        mov tmpSI,si
        ret  ;возврат из процедуры
PutChar endp      ;конец
процедуры
PutCharEnd  procfar
        mov si,tmpSI
        mov buffer[si],'$'
        ret
PutCharEnd  endp
code      ends
end
```

```
;mac.inc
OutStr  macro  str
;Вывод строки на экран.
;На входе - идентификатор начала выводимой
строки.
;Строка должна заканчиваться символом '$'.
;На выходе- сообщение на экране.
      push    ax
      mov     ah,09h
      lea dx,str
      int  21h
      pop     ax
endm
GetChar    macro
;Ввод символа с клавиатуры.
;На выходе - в al введённый символ.
      push    ax
      mov     ah,01h
      int  21h
      pop     ax
endm
```

```
OutChar macro
;Вывод символа на экран.
;На входе - в dl выводимый символ.
      push    ax
      mov     ah,02h
      int  21h
      pop     ax
endm
clear_r macro  rg
;очистка регистра rg
      xor rg,rg
      endm
conv_16_2macro
;макрос преобразования символа
шестнадцатеричной цифры
;в ее двоичный эквивалент в al
      subdl,30h
      cmp     dl,9h
      jle  $+5
      subdl,7h
      endm
```

```
GetStr    macro    buf,max_len
;ввод строки произвольной длины (функция 0ah int 21h)
;на входе:
;buf - адрес строки куда будет помещен ввод
;max_len - максимальная длина вводимой строки
;на выходе - введенная строка по адресу buf
;al - длина введенной строки
      push      es
      push      dx
      push      cx
      xor  cx,cx
      mov buf,max_len
      mov ah,0ah
      lea  dx,buf
      int   21h
      mov al,buf+1
      mov cl,al ;длина введенной строки в al
;сдвиг al на два байта влево:
      push      ds
      push      es
      lea  si,buf+2
      lea  di,buf
rep  movsb
      pop cx
      pop dx
      pop es
GetStr    endm
```

```
init_ds    macro
;макрос настройки ds на сегмент данных
        mov ax,data
        mov ds,ax
        xor  ax,ax
        endm
delay      macro    time
        local       ext,iter
;макрос задержки. На входе - значение
;переменной задержки (в мкс).
        push       cx
        mov cx,time
ext:  push       cx
        mov cx,5000  ;это значение можно поменять, исходя из
;производительности процессора.
iter:  loop iter
        pop  cx
        loop ext
        pop  cx
        ENDM
Exit macro
;Выход из программы.
        mov ax,4c00h
        int   21h
endm
```

Передача аргументов через стек

При передаче управления процедуре МП автоматически записывает в вершину стека два (для процедур типа NEAR) или четыре (для процедур типа FAR) байта - адрес возврата в вызывающую программу.

Для осуществления произвольного доступа к данным в стеке архитектура МП имеет специальный регистр BP (Base Point – указатель базы). Так же как и для регистра SP, использование BP автоматически предполагает работу с сегментом стека.

```
MASM
MODEL SMALL
proc_1 PROC NEAR  ;”близкая” процедура (NEAR)
с n аргументами
;начало пролога
     PUSH   BP
     MOV    BP,SP
;конец пролога
     MOV    AX,[BP+4] ;доступ к аргументу arg_n
для NEAR-процедуры
     MOV    AX,[BP+6] ;доступ к аргументу arg_{n-1}
     …      ;команды процедуры
;подготовка к выходу из процедуры
;начало эпилога
     MOV    SP,BP  ;восстановление SP
     POP    BP ;восстановление значения старого
BP
                ;до входа в процедуру
     RET        ;возврат в вызывающую программу
;конец эпилога
proc_1 ENDP
```

```
.CODE
main   PROC
     MOV   AX,@data
     MOV   DS,AX
…
PUSH arg_1       ;запись в стек 1-го аргумента
PUSH arg_2       ;запись в стек 2-го аргумента
…
PUSH arg_n       ;запись в стек n-го аргумента
CALL  proc_1          ;вызов процедуры proc_1
;действия по очистке стека после возврата из
процедуры
…
m1:
     _exit
main   ENDP
END   main
```

Использование директив EXTRN и PUBLIC

При передаче управления процедуре МП автоматически записывает в вершину стека два (для процедур типа NEAR) или четыре (для процедур типа FAR) байта - адрес возврата в вызывающую программу.

Для осуществления произвольного доступа к данным в стеке архитектура МП имеет специальный регистр BP (Base Point – указатель базы). Так же как и для регистра SP, использование BP автоматически предполагает работу с сегментом стека.

```
;prg4.asm
;Вызывающий модуль
include  mac.inc
extrn    my_proc2:far
public   per1,per2
stk  segment     stack
     db  256 dup (0)
stk  ends
data     segment
per1     db  '1'
per2     db  '2'
data     ends
code     segment
main     proc     far
assume
cs:code,ds:data,ss:stk
     mov      ax,data
     mov      ds,ax
     call my_proc2
     exit
main     endp
code     ends
end main
```

```
;prg5.asm
;Вызываемый модуль
include     mac.inc
extrn
per1:byte,per2:byte
public  my_proc2
code    segment
my_proc2 proc    far
assume    cs:code
;вывод символов на
экран
      mov    dl,per1
      OutChar
      mov    dl,per2
      OutChar
      ret
my_proc2 endp
code    ends
end
```