Section Preview

❒ **Writing Reentrant Programs**

- ♦ **Module attributes**
- ♦ **Writing Reentrant Programs**
- ♦ **Reentrant Save Area Chaining**
- ♦ **Reentrant I/O**
- ♦ **Reentrant Processing**
- ♦ **Sample Reentrant Program**
- ♦ **Using list and execute form of macros**

Module Attributes

❒ **A module is ...**

Reusable

♦ **If it initializes itself on entry, or re-initializes itself on exit**

✗ Thus we can reuse a copy of this module already in memory, without having to get a fresh copy from the library on disk

✗ If multiple tasks LINK to or ATTACH this program simultaneously, the first requestor will get control while the others will wait; when the first requestor completes the next requestor can use the subroutine

Module Attributes, 2

❒ **A module is ...**

Reenterable (also Reentrant)

- **If multiple tasks can LINK to or ATTACH the same program simultaneously, and only one copy is needed in memory**
  - ✗ All tasks can share the code simultaneously
  - ✗ This implies the module never modifies itself (or if it does, it uses serialization techniques when modification is being done)
  - ✗ Many system routines are reenterable (OPEN, CLOSE, access method modules, and so on)

❒ **There are three ways to make a module reenterable:**

- **Dynamically obtain storage to hold all program data areas that need to be changed**
- **Use instructions or system services to serialize code when data areas in the module need to be changed**
- **A combination - some dynamic storage, some serialization**

❒ **For an application program, the dynamic storage approach is the best: serialization poses serious performance risks that should only be undertaken in code written to extend the operating system**

Specifying Module Attributes

❒ **You mark a module as being reusable by specifying REUS as a Program Binder parm:**

```
//LKED      EXEC  PGM=IEWL,PARM='...REUS...'
```

❒ **You mark a module as being reenterable by binding with the RENT attribute:**

```
//LKED      EXEC  PGM=IEWL,PARM='...RENT...'
```

Notes

- **The default attributes are not-REUS and not-RENT**
- **RENT implies REUS, but not the other way around**
- **Marking a program RENT or REUS does not make it so**
  - ✗ You must code the program so it is actually reusable or reenterable
  - ✗ Specifying the Assembler option RENT requests the Assembler to flag any place it appears your program is not reenterable (although you can fool the Assembler)

Writing Reentrant Programs

❒ **Programs are made reentrant primarily by using two programming guidelines**

Use registers instead of storage areas

- **Each requester of a module always provides its own register save area, so each requester will see only its own registers**

When you must use storage locations ...

- **Use a register format of GETMAIN to obtain storage outside of your program and work with fields in the GETMAINed area**
  - ✗ The system ties GETMAINed areas to the task making the request
  - ✗ When a reentrant program is dispatched, the system ensures any GETMAINed storage areas it uses are the ones tied to the task that the program is currently running under

❒ **CAUTION**

- **Some macros generate code that modifies storage**
- **For these macros you might have to: code a List form of the macro, move the generated list code into a GETMAINed area, and issue the Execute form of the macro, pointing to the list in the GETMAINed area**

Writing Reentrant Programs, 2

❒ **At Assembly time, specify the Assembler parm RENT**

- ♦ **The Assembler will check for obvious violations of reentrancy, but it can be fooled**

❒ **At bind time, specify the Program Binder parm RENT**

❒ **At run time, if the module is not truly reentrant, the results may be any of these ...**

- ♦ **The program runs fine, with no apparent flaws**
  - ✗ There may be a problem that will not show up until two separate tasks actually invoke the program simultaneously
  - ✗ The program might be reentrant in a way that is trivial
  - ✗ The program might work fine until it is placed into the LPA
- ♦ **The program runs with completion code zero but the results are not right**
  - ✗ Switches and counters may be used by conflicting tasks so that only the resulting values are out of sync
- ♦ **Abends may occur**

Reentrant Save Area Chaining

- A reentrant program is no different from other programs in one respect: it must provide a 72-byte register save area
  - Recall this save area is used by subroutines or system services, for storing the program's own registers while the subroutine or system service does its work
- Clearly, the register save area will have to be in a GETMAINed area, or the module will not be reentrant
  - Here is one possible approach to coding reentrant save area chaining

```
MYPROG   CSECT
         STM     14,12,12(13)
         USING   MYPROG,12
         LR      12,15
         L       2,0(1)    save pointer to parms
         GETMAIN R,LV=72   get save area
         ST      13,4(1)
         LR      13,1
         .
         .
         LR      1,13
         L       13,4(13)
         FREEMAIN R,LV=72,A=(1)
         RETURN  (14,12),,RC=0
         .
         .
         END     MYPROG
```

Reentrant I/O

- The DCB macro generates a control block that describes a data set
- At OPEN time, the OPEN routines put the address of the correct access method modules into the DCB
  - That is, the DCB is modified!
  - Clearly, we have to do some special processing
- Similarly, if you GET a record into an area in your program you have modified your program
  - Again, we need to do some special processing
  - Here is an example of a reentrant program that does I/O

```
            GETMAIN  R,LV=DCBSZ   get storage for DCB
            LR       3,1
            MVC      0(DCBSZ,3),INDCB
            GETMAIN  R,LV=240     get storage for a
*                                 record area
            LR       4,1
            OPEN     ((3),INPUT)
            GET      (3),(4)
            .
            .
INDCB       DCB      DSORG=PS,...
ENDINDCB    EQU      *
DCBSZ       EQU      ENDINDCB-INDCB
```

Reentrant Processing

- **Clearly, any processing your program does (calculations, editing, moving, translating, etc.) will have to use GETMAINed areas for work areas**
  - **The use of DSECTs can greatly simplify this work**
- **The following pages demonstrate a complete reentrant program that reads a file and creates a report**
  - **This example assumes the program takes the default AMODE and RMODE of 24 / 24**
    - ✗ A discussion of doing I/O running AMODE 31 is contained in a separate paper

Sample Reentrant Program

```
RENTER   CSECT
R1       EQU     1
R2       EQU     2
R3       EQU     3
R4       EQU     4
R5       EQU     5
R6       EQU     6
R7       EQU     7
R12      EQU     12
R13      EQU     13
R14      EQU     14
R15      EQU     15
         STM     R14,R12,12(R13)
         USING   RENTER,R12
         LR      R12,R15
         L       R2,0(R1)    save pointer to parms
         GETMAIN R,LV=72     get save area
         ST      R13,4(R1)
         LR      R13,R1
*   put input file DCB into a GETMAINed area
         GETMAIN R,LV=DCBSZ1
         LR      R3,R1
         MVC     0(DCBSZ1,R3),INFILE
*   put output file DCB into a GETMAINed area
         GETMAIN R,LV=DCBSZ2
         LR      R4,R1
         MVC     0(DCBSZ2,R4),OUTFILE
```

Sample Reentrant Program, continued

```
*********************************************
*   get storage for input record and establish
*   addressability for input record DSECT
*********************************************
         GETMAIN R,LV=100
         LR      R5,R1
         USING   INDSECT,R5
*
*********************************************
*   get storage for output record and establish
*   addressability for output record DSECT.
*   Initialize area to blanks
*********************************************
         GETMAIN R,LV=100
         LR      R6,R1
         USING   OUTDSECT,R6
         MVI     OUT_REC,C' '
         MVC     OUT_REC+1(99),OUT_REC
*
*********************************************
*   get storage for doubleword work area and
*   establish addressability to a work DSECT
*********************************************
         GETMAIN R,LV=8
         LR      R7,R1
         USING   WORKSECT,R7
```

Sample Reentrant Program, continued

```
*********************************************
*   Open files, put out title line
*********************************************
          OPEN   ((R3),,(R4),(OUTPUT))
          PUT    (R4),RPT_TITLE
*
*********************************************
*   Main logic
*********************************************
LOOP      DS     0H
          GET    (R3),INREC
          MVC    OUT_PART,INPART#
          MVC    OUT_DESC,INDESCR
          MVC    OUT_UNPR,EDPAT_PRICE
          ED     OUT_UNPR,INUNPRCE
          MVC    OUT_QOH,EDPAT_QTY
          ED     OUT_QOH,INQTYHND
          MVC    OUT_QORD,EDPAT_QTY
          LH     R2,INQTYORD
          CVD    R2,DBLWRD
          ED     OUT_QORD,DBLWRD+5
          MVC    OUT_REOR,EDPAT_QTY
          LH     R2,INREORDR
          CVD    R2,DBLWRD
          ED     OUT_REOR,DBLWRD+5
          PUT    (R4),OUT_REC
          B      LOOP
```

Sample Reentrant Program, continued

```
*********************************************
*   When done, close files and clean up
*   GETMAINed areas
*********************************************
DONE      DS     0H
          CLOSE  ((R3),,(R4))
          LR     R1,R7
          FREEMAIN R,LV=8,A=(R1)
          LR     R1,R6
          FREEMAIN R,LV=100,A=(R1)
          LR     R1,R5
          FREEMAIN R,LV=100,A=(R1)
          LR     R1,R4
          FREEMAIN R,LV=DCBSZ2,A=(R1)
          LR     R1,R3
          FREEMAIN R,LV=DCBSZ1,A=(R1)
          LR     R1,R13
          L      R13,4(R13)
          FREEMAIN R,LV=72,A=(R1)
          LM     R14,R12,12(R13)
          SR     R15,R15
          BR     R14
```

Sample Reentrant Program, continued

```
*********************************************
*   Constants and data areas
*********************************************
*
RPT_TITLE DC  CL32'    Inventory Status Report For'
          DC  CL68'Novelty Products Division'
*
EDPAT_PRICE  DC  X'40206B2021204B202020'
EDPAT_QTY    DC  X'4020206B202120'
*
INFILE   DCB   DSORG=PS,MACRF=GM,DDNAME=IN,       X
               EODAD=DONE
DCBSZ1   EQU   *-INFILE
*
OUTFILE  DCB   DSORG=PS,MACRF=PM,LRECL=100,       X
               DDNAME=OUT,BLKSIZE=0,RECFM=FB
DCBSZ2   EQU   *-OUTFILE
```

Sample Reentrant Program, continued

```
*************************************************
*   DSECTs
*************************************************
*
INDSECT  DSECT
INREC    DS     0CL100
INPART#  DS     CL9
INDESCR  DS     CL30
         DS     CL5
INUNPRCE DS     PL4
INQTYHND DS     PL3
         DS     C
INQTYORD DS     H
INREORDR DS     H
         DS     CL44
*
```

Sample Reentrant Program, continued

```
OUTDSECT DSECT
OUT_REC  DS      0CL100
         DS      C
OUT_PART DS      CL9
         DS      CL3
OUT_DESC DS      CL30
         DS      CL3
OUT_UNPR DS      CL10
         DS      CL3
OUT_QOH  DS      CL7
         DS      CL3
OUT_QORD DS      CL7
         DS      CL3
OUT_REOR DS      CL7
         DS      CL14
*
WORKSECT DSECT
DBLWRD   DS      D
*
         END     RENTER
```

Performance Concerns

- **After you get your program coded and tested, consider spending some time consolidating all your GETMAINed areas into fewer chunks (ideally: one) to gain performance**

- **A single DSECT can be used to include multiple areas**
  - **For example, INREC, OUT_REC, and DBLWRD could all be under one DSECT**
    - ✗ Requiring only one USING and one register

```
        USING  SDSECT,R5
```

Performance Concerns, 2

❒ **Multiple DSECTs can be used over a single GETMAINed area**

- **For example, GETMAIN for 208 bytes and assign the first 100 to INREC, the next 100 to OUT_REC, and the last 8 to DBLWRD**

  ✗ Requires a register and USING for each DSECT, as before, just a different approach for initializing the base registers

```
        GETMAIN R,LV=208
        LR      R5,R1
        LA      R6,100(R1)
        LA      R7,200(R1)
        USING   INDSECT,R5
        USING   OUTDSECT,R6
        USING   WORKSECT,R7
```

❒ **Or, a little nicer, use dependent USINGs:**

```
        GETMAIN R,LV=208
        LR      R5,R1
        USING   INDSECT,R5
        USING   OUTDSECT,INDSECT+L'INREC
        USING   WORKSECT,OUTDSECT+L'OUTREC
```

Using List And Execute Form Of Macros

❒ **Just to demonstrate coding a reentrant program that uses some macros that require list and execute forms to remain reentrant, here is an example that uses CALL and WTO macros**

- **The program, WTOPARM, simply:**
  - ✗ Issues an "Entered" message (WTO)
  - ✗ CALLs a subroutine, LOWERC, passing any data from the JCL PARM field plus a work areas
  - ✗ Issues a "Leaving" message (WTO)

- **The subroutine, LOWERC, also reentrant, does this:**
  - ✗ Issues a message with the orignal PARM text (WTO)
  - ✗ Foreces the text to be lowercase after the first letter
  - ✗ Issues a message with the modified PARM text (WTO)

Using List And Execute Form Of Macros, 2

- First the mainline:

```
*process compat(macrocase),rent     allows lower case
wtoparm   CSECT
wtoparm   amode   31
wtoparm   rmode   any
*  Copyright  (C)  2005 by Steven H. Comstock
          stm     14,12,12(13)
          lr      12,15
          l       2,0(,1)    save address of parms
          using   wtoparm,12
          getmain  r,lv=72   for save area
          st      13,4(1)
          lr      13,1
*  get storage for work area
          getmain  r,lv=my_size
          lr      3,1 for work area
          using   workarea,3
*  populate gotten storage with WTO and CALL
*  parameter lists
          mvc     wto_er(wto_size),wto_base
          mvc     call_er(call_size),caller
```

Using List And Execute Form Of Macros, 3

❒ First the mainline, continued:

```
*  put out entry message
           la     5,in_msg_er
           wto    text=(5),mf=(e,wto_er)
*  call subroutine, passing parm from JCL
*     and work area
           call   lowerc,((2),outer),mf=(e,call_er)
*  put out exit message
           la     5,out_msg_er
           wto    text=(5),mf=(e,wto_er)
*  free work area storage
           lr     1,3
           freemain  r,lv=my_size,a=(1)
*  free save area storage
           lr     1,13
           l      13,4(1)
           freemain  r,lv=72,a=(1)
*  return to z/OS
           lm     14,12,12(13)
           sr     15,15
           br     14
```

Using List And Execute Form Of Macros, 4

❒ First the mainline, continued:

```
in_msg_er  ds   0cl18
           dc    h'16'
           dc    cl16'Entered wtoparm.'
out_msg_er ds   0cl18
           dc    h'16'
           dc    cl16'Leaving wtoparm.'
caller     call  ,(0,0,0,0),mf=l
wto_base   wto   text=,routcde=(11),mf=l
workarea   dsect
call_er    call  ,(0,0,0,0),mf=l
call_size  equ   *-call_er
wto_er     wto   text=,routcde=(11),mf=l
wto_size   equ   *-wto_er
outer      ds    cl102
my_size    equ   *-call_er
           end   wtoparm
```

Notes

- The CALL is set up so you can use the list form for any execute form that passes up to four parameters
- Notice how the size of various areas is captured using EQUs
- Notice how the DSECT area is populated from the models in the non-DSECT area

Using List And Execute Form Of Macros, 5

❒ Now, the subroutine:

```
*process compat(macrocase),rent
lowerc    CSECT
lowerc    amode  31
lowerc    rmode  any
*   Copyright (C) 2005 by Steven H. Comstock
          stm    14,12,12(13)
          lr     12,15
          lr     2,1         save address of parms
          using  lowerc,12
          getmain r,lv=72 for save area
          st     13,4(1)
          lr     13,1
          lm     4,5,0(2)
*  c(4) = a(in);c(5) = a(out)
          getmain r,lv=wto_size
          lr     6,1         for wto area
          mvc    0(wto_size,6),model_wto
*  display original text
          wto    text=(4),routcde=(11),mf=(e,(6))
*  adjust length to move (data + len field)
          lh     3,0(3)
          la     3,1(3)
*  copy whole input area to output area
          ex     3,moveit
```

Using List And Execute Form Of Macros, 6

❒ Now, the subroutine, continued:

```
*  subtract 2 to ignore length prefix in translate
           bctr   3,0
           bctr   3,0
           bctr   3,0    plus, skip first byte
*  translate input to lower case
           ex     3,trans_it
*  display translated text
           wto    text=(5),routcde=(11),mf=(e,(6))
*  free gotten storage areas and return
           lr     1,6
           freemain r,lv=wto_size,a=(1)
           lr     1,13
           l      13,4(1)
           freemain r,lv=72,a=(1)
           lm     14,12,12(13)
           sr     15,15
           br     14
*  model statements for EX statements
moveit     mvc    0(0,5),0(4)
trans_it   tr     3(0,5),trtable
```

Using List And Execute Form Of Macros, 7

❒ Now, the subroutine, continued:

```
trtable  ds     0cl256
         dc     x'000102030405060708090A0B0C0D0E0F'
         dc     x'101112131415161718191A1B1C1D1E1F'
         dc     x'202122232425262728292A2B2C2D2E2F'
         dc     x'303132333435363738393A3B3C3D3E3F'
         dc     x'404142434445464748494A4B4C4D4E4F'
         dc     x'505152535455565758595A5B5C5D5E5F'
         dc     x'606162636465666768696A6B6C6D6E6F'
         dc     x'707172737475767778797A7B7C7D7E7F'
         dc     x'808182838485868788898A8B8C8D8E8F'
         dc     x'909192939495969798999A9B9C9D9E9F'
         dc     x'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'
         dc     x'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'
         dc     x'C0818283848586878889CACBCCCDCECF'
         dc     x'D091929394959697989A9B9C9D9DDEDF'
         dc     x'E0E1A2A3A4A5A6A7A8A9EAEBECEDEEEF'
         dc     x'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'

model_wto wto    text=,routcde=(11),mf=l
wto_size  equ    *-model_wto

          end    lowerc
```

Further Explorations

- **Topics of interest regarding the use of virtual storage**
  - **STORAGE macro (OBTAIN and RELEASE)**
  - **CPOOL macro (work with cell pools and cells)**
  - **DSPSERV macro (create, delete, control Data Spaces and Hiperspaces)**
  - **HSPSERV macro (read from and write to a Hiperspace)**
- **Other advanced topics**
  - **Using RSECTs to define read-only control sections**
  - **Address space creating, control, deletion**
  - **Use of linkage stacks**