Looking at Virtual Storage

BY SAM GOLOB

Many of us would write more code if we could only see what we were writing about. Back when I was an application programmer, if I had to write a program to manipulate a file, I would obtain a sample production file and copy it as a test file. Then I'd look at it very carefully before attempting to write my code since it's always important to see the data before writing a program for it.

I also remember a time when, as a systems programmer, I enhanced the TAPEMAP program from File 299 of the CBT MVS Tape. TAPEMAP is very careful to "feel" the data at the beginning of each tape file in order to classify it as IEBCOPY format, IEBUPDTE, SMPPTFIN, etc. Therefore, I needed the TAPESCAN program from File 102 of the CBT tape or the DITTO tape functions to give me a byte-by-byte detailed description of all the tape files to be studied in hex. I had to "look"in order to be able to program.

Now, here's the $64 question: When you're a systems programmer writing code that accesses system control blocks in virtual storage, how can you get a good look at the data you're writing about? Remember that you have to look at actual virtual storage that is running in your address space or in another address space, and it helps if you can follow control block chains.

There are a few vendor tools to do this such as the XDC Debugger from Cole Software and TSO TEST; however, there's also a free tool that can be obtained and used by anyone. This is the extremely clever LOOK program from File 264 of the CBT MVS Utilities Tape. The original LOOK program, which was once part of a free software product called DCMS, was enhanced by Guy Albertelli. This LOOK program or any similar product, is an

Figure 1: MYID

MYID is a simple TSO command that finds the user ID of the invoker and displays it on the tube. This command gets the user ID from the PSCBUSER field of the TSO Protected Step Control Block (PSCB). This field is obtained by running a control block chain starting from the CVT, as illustrated by the code. Next, we'll use the LOOK program to actually see the storage at each stage.

```
*      M Y I D   P R O G R A M   -   A   T S O   C O M M A N D
*
*      TSO COMMAND PROCESSOR TO DISPLAY THE USERID OF THE INVOKER.
*
*     ——  REGISTER EQUATES ELIMINATED FOR BREVITY  ————
*
MYID        CSECT
            STM   R14,R12,12(R13)          SAVE CALLER'S REGISTERS R14 THRU R12
            LR    R12,R15                  LOAD ENTRY POINT INTO BASE REGISTER
            USING MYID,R12                 TELL THE ASSEMBLER, R12 IS THE BASE
            LR    R15,R13                  SAVE CALLER'S SAVEAREA ADDRESS
            LA    R13,SAVE                 POINT R13 TO OUR SAVEAREA ADDRESS
            ST    R15,SAVE+4               STORE HIS SAVEAREA INTO MINE + 4
            ST    R13,8(,R15)              STORE MINE INTO HIS SAVEAREA + 8
RUNCHAIN    L     R3,16                    POINT TO CVT. ADDR IS IN LOW STORAGE
            L     R3,0(,R3)                POINT TO TCB/ASCB WORDS, "0" OFF CVT
            L     R3,4(,R3)                POINT TO TCB, "4" OFF TCB/ASCB WORDS
            L     R3,X'B4'(,R3)            POINT TO JSCB. X'B4' OFF CURRENT TCB
            L     R3,X'108'(,R3)           POINT TO PSCB. X'108' OFF THE JSCB
            MVC   MSGLINE+13(7),0(R3)        MOVE USERID IN FROM 0 OFF THE PSCB
            TPUT  MSGLINE,L'MSGLINE        DISPLAY THE WHOLE MESSAGE ON THE TUBE
RETURN      DS    0H
            L     R13,SAVE+4               RELOAD CALLER'S SAVEAREA POINTER
            LM    R14,R12,12(R13)          RELOAD THE CALLER'S REGISTERS
            BR    R14                      RETURN TO CALLER
SAVE        DC    18F'0'                   DEFINE MY SAVEAREA - 18 FULLWORDS
MSGLINE     DC    CL20'MY USERID IS          '       DEFINE LINE FOR MESSAGE
            END
```

Figure 2: The LOOK Screen With the First Page of the CVT Formatted

You get to the CVT from the pointer at virtual location X'10' or Decimal 16. In LOOK parlance, the command is J10. You can see that LOOK uses the hex value.

```
LOOK COMMAND - DISPLAY VIRTUAL MEMORY       DISPLAY ASID= 0029
ENTER CMD -
LAST CMD  - J10
                              00FDA068  CVT
 TCBP    00000218    0EF00     00FE5E90  LINK     00FD511C  AUSCB     00FD7D48
 BUF     00000000    XAPG      00FF80C4  QVL00    00FF2ECE  PCNVT     00FFC114
 PRLTV   00FFBF44    LLCB      0155ECD8  LLTRM    81162940  XTLER     00FEAEF8
 SYSAD   00F4A080    BTERM     00FF5030  DATE     0097334F  MSLT      00FD5908
 ZDTAB   00F41000    XITP      00FFA5E8  0EF01    00FE60B8  VSS       0000
 VPSM    0000  SVDCB    00FD5124  TPC      00FD3308  RS05C    0000  ICPID    0000
 RS060   40C3E5E3    CUCB      00FD5AA0  QTE00    00FDE90A  QTD00     00FDE92A
 STB     00F4C580    DCB       9B  DCBA     FD7E18  SV76M    00000000
 IXAVL   00FE1400    NUCB      00000000  FBOSV    8150A278  0DS       00FEB9B8
 ECVT    0155EE18    DAIRX     82F11000  MSER     00FD5908  OPT01     00FEEA18
 TVT     00CA6260    SV76C     00000000  MZ00     7F  1EF00    00000000
 QOCR    00000000    QMWR      00FD7D10  SNCTR    0000  OPTA     A3    OPTB     20
 QCDSR   00FE66E0    QLPAQ     00FD5158  ENFCT    00FDA650  SMCA      80FC2B20
 ABEND   00FD3210    USER      00000000  MDLDS    00000000  TSCE      00000000
 PATCH   00FD4978    RMS       0142E190  SPDME    01F93B6C  OSCR1     00FFB598
 GTFST   00  GTFA     FCEC20   TCMFG    00  AQAVB    000000  RS0F4    00000000

1= HELP       2=             3= END         4=             5= REPEAT      6=
7= BACKWARD   8= FORWARD     9= HIST BWD    10= HIST FWD   11=            12=
```

Figure 3: Unformatted Storage at the Location of the TCB Pointers for Your Address Space

If LOOK recognizes a control block and formats it, you can turn off the formatting with the command "ONULL."
If you want to purposely format some storage as if it were a CVT, you would use the command "OCVT." You get the idea.

```
LOOK COMMAND - DISPLAY VIRTUAL MEMORY       DISPLAY ASID= 0029
ENTER CMD -
LAST CMD   - J+0

00000218    >009B81F8     009B81F8     00FD3600     00FB6D00    * >...8...8....._.*
00000228     00000000     00000000     00000000     00000000     *................*
00000238     00000000     00000078     00000000     00000000     *................*
00000248     00000000     7F748950     040C0000     00DF1DF2     *....".&.......2*
00000258     040C0000     8003986C     AD000950     AD000950     *.......%...&...&*
00000268     AD040950     AD000950     AD040950     00000000     *...&...&...&....*
00000278     00000704     00000000     00FD13E8     00000000     *...........Y....*
00000288     00FD13F0     00000000     00000000     00000000     *...0............*
00000298     00000000     00000000     00000000     00000000     *................*
000002A8     00FD13F8     00000000     00FD1430     00000000     *...8............*
000002B8     00000000     00000000     00000000     00FD1440     *................ *
000002C8     00000000     00000000     00FD1438     00FD1448     *................*
000002D8     00FD1450     00000000     00000000     00000000     *...&............*
000002E8     00000000     00000000     00000000     00000040     *................ *
000002F8     00000000     00FEBAD0     00000000     00000000     *................*
00000308     5FB1EE40     00040000     0000000      00000000     *¬.. ............*

1= HELP       2=             3= END         4=             5= REPEAT      6=
7= BACKWARD   8= FORWARD     9= HIST BWD    10= HIST FWD   11=            12=
```

Figure 4: A Storage Description of the First TCB Formatted as a TCB by the LOOK Program

```
LOOK COMMAND - DISPLAY VIRTUAL MEMORY       DISPLAY ASID= 0029
ENTER CMD -
LAST CMD   - J+4
                         009B81F8  TCB
 FRS0      0000000000000000  FRS2     0000000000000000  FRS4     0000000000000000
 FRS6      0000000000000000  RBP      009B8110    PMASK    00   PIEA     000000
 DEB       00000000  TIO      009C8000  CMPF      94   CMPC     0C4000   ABF      40
 TRNB      000000   MSSB     7547A8   PKF       80   FLGS1    00   FLGS2    00
 FLGS3     00   FLGS4    00   FLGS5    00 LMP       FF   DSP      FE   LLS      00000000
 JLB       0000BF14  PURGE    00   JPQB     000000   GRS0      00E23A78
 GRS1      00000009  GRS2     00000318  GRS3      00000003   GRS4     0007E9C0
 GRS5      00000000  GRS6     0007F100  GRS7      00023E90   GRS8     00081DE0
 GRS9      05A5F7D8  GRS10    0007E518  GRS11     85A5E7D8   GRS12    846C3D18
 GRS13     0007E518  GRS14    85A5EAA4  GRS15     00000000   FSAB     07DFB0
 TCB       00000000  TME      00000000  JSTCA     9FD178   NTC      00000000
 OTC       009D9298  LTC      00000000  IQE       00000000   ECB      0007312C
 TSFLG     00   STPCT    00   TSLP     00   TSDP     00   RD       7FF16EE4
 AE        00000000  NSTAE    00   STABB    9FF410   TCTGF     80   TCTB     9D9D58
 USER      00000000  NDSP0    00   NDSP1    00   NDSP2    00    NDSP3    00
 MDIDS     00000000  RECDE    00   JSCBB    9FD234   SSAT      00FD35B8

1= HELP       2=             3= END         4=             5= REPEAT      6=
7= BACKWARD   8= FORWARD     9= HIST BWD    10= HIST FWD   11=            12=
```

Figure 5: The Unformatted Storage Pointed to by the Address at X'B4' off the Beginning of the TCB in Figure 4

This is the JSCB (Job Step Control Block). From the formatted TCB display shown in Figure 4, we can also get here with a label Link command, "LJSCBB."

```
LOOK COMMAND - DISPLAY VIRTUAL MEMORY       DISPLAY ASID= 0029
ENTER CMD -
LAST CMD   - J+B4

009FD234    >00000000     00000000     00000000     00000000    * >...............*
009FD244     00000000     009FD2D0     00000000     7FFFED60     *......K.....".-*
009FD254     009FDE88     00000000     00000000     00000000     *................*
009FD264     00000000     00000040     7FFFC02C     7FFDDEB0     *....... "...".../*
009FD274     00000000     E3C3C240     00000000     00000000     *....TCB ........*
009FD284     7FFDD370     0000FFFF     00048000     00000000     *".L.............*
009FD294     00064E78     7FFD6DC0     00000000     7FF16EF4     *..+.".._....."1.4*
009FD2A4     7FFDD7F0     00000000     00000000     7FFF5138     *".P0........"...*
009FD2B4     00000000     5A8DF658     00006E88     00000003     *....!.6.........*
009FD2C4     00000000     00000000     00000000     00000000     *................*
009FD2D4     00000000     00000000     0000000C     00000000     *................*
009FD2E4     00000000     00000000     00000000     00000000     *................*
009FD2F4     00000000     00000000     00000000     00000000     *................*
009FD304     009FDE88     00000000     009D9528     00000000     *................*
009FD314     00000000     01000000     00000000     00000000     *................*
009FD324     00000000     009DE940     00000000     00000000     *......Z ........*

1= HELP       2=             3= END         4=             5= REPEAT      6=
7= BACKWARD   8= FORWARD     9= HIST BWD    10= HIST FWD   11=            12=
```

absolute must to use if you're writing system-level code and need to see the data in detail.

Now, here's the $64 question: When you're a systems programmer writing code that accesses system control blocks in virtual storage, how can you get a good look at the data you're writing about?

As I've mentioned, you can find LOOK on the CBT MVS Utilities Tape, an independently produced tape that is distributed by several sources including NaSPA. The tape, which is of immense value to MVS systems programmers, may be freely copied. When you need to do a job and don't know how to do it, the CBT Tape is one of the first places to search to find out if someone else has done a similar job already. Most of the software on the CBT Tape is in source form, and you can study it and modify it to your heart's content. Therefore, you can get your job done and learn to code better than you could before in assembler, PL/I, or in whatever language the software is written.

Now let's see how the LOOK program can show us in-storage data and format it automatically as a system control block. Then we'll navigate through storage from control block to control block, from pointer to pointer and from there to the data.

My friend and teacher Jeff Broido told me early in my training that whenever he writes system code to access control blocks he always views what he's doing by using the LOOK program.

What kind of program is LOOK? LOOK is a TSO command that is designed to run interactively in full-screen mode under TSO. LOOK will show you chunks of virtual storage in your own address space, in hex and EBCDIC. This gives you access to common storage as well. If LOOK is running authorized, it is capable of showing similar private storage in any other address space. LOOK does this by issuing an SRB to go cross-memory to retrieve the block of storage from the other address space. A LOOK user goes from one full-screen of

Figure 6: An Unformatted PSCB

The PSCB is X'108' off the previous location, which is the JSCBPSCB field of the JSCB. We could not get to this location with a label Link command, because the previous screen was not formatted as the JSCB control block. We are looking at the storage that is pointed to by the address at that location. Lo and behold, the PSCB can be seen in storage. The PSCBUSER field of the PSCB is seven bytes long, starting from the beginning of the PSCB control block. And it's got our user ID there. In our program, we just have to move these seven bytes into the TPUT display to the terminal and we're done. It pays to see what you are doing!

```
LOOK COMMAND - DISPLAY VIRTUAL MEMORY       DISPLAY ASID= 0029
ENTER CMD -
LAST CMD  - J+108

00005F80    >E2C1C7D6    D3D6C207    E2E8E2C1    D3D3C4C1 *   >SAGOLOB.SYSALLDA*
00005F90     A2000000    AFA3EAB7    05B5FC00    00000000     *.................*
00005FA0     0000006F    00000000    00000000    00000000     *...?............*
00005FB0     00006EC0    00006FC8    00380000    00000C00     *......?H........*
00005FC0     00000000    00000000    00000000    00000004     *.................*
00005FD0     00000000    00000000    00000000    00000000     *.................*
00005FE0     00000000    00000000    00000000    00000000     *.................*
00005FF0     00005FF8    00000000    C9D2D1D3    D1F14040     *..¬8....IKJLJ1  *
00006000     00000000    00000000    00000000    00000000     *.................*
00006010     00000000    00000000    00000000    00000000     *.................*
00006020     00000000    00000000    00000000    00000000     *.................*
00006030     00000000    00000000    00000000    00000000     *.................*
00006040     00000000    00000000    00000000    00000000     *.................*
00006050     00000000    00000000    00000000    00000000     *.................*
00006060     00000000    00000000    00000000    00000000     *.................*
00006070     00000000    00000000    00000000    00000000     *.................*

1= HELP       2=            3= END        4=            5= REPEAT     6=
7= BACKWARD   8= FORWARD    9= HIST BWD   10= HIST FWD  11=           12=
```

Figure 7: HELP Screen for the LOOK Command

This will remind the user of what can be done using LOOK. There's really a lot of flexibility if you study this command set.

```
LOOK COMMAND - DISPLAY VIRTUAL MEMORY     DISPLAY ASID= 0029
ENTER CMD -
LAST CMD  - HELP

LOOK is a real time core display and formatting program. It also has the
capability of displaying memory in any address space (if authorized).

The valid commands are:
lexp      24 bit indirect                  | Jexp     31 bit indirect
>         Forward                          | <        Backward
=sym      Define current address as "sym"  | ,sym     Redisplay core at "sym"
M0/M1     Flip between top and center      | Lname l  ndirect thru control block field
Ocb       Format as "cb" control block     | R        Refresh displayed storage
          "cb" may be NULL to show as hex  |

where     'exp' is of the form:
   <+/->hhhh<+/-hhhh<+/-hhhh...>>
and 'hhhh' is a 1 to 8 digit hex number.
```

data to another; two PFkeys enable the user to go backward or forward to the various screens. From any screen of data, you can explore new storage as desired. Once you learn the subcommands of LOOK, you'll come to appreciate its great flexibility.

Let's see how LOOK works by actually doing something. I once wrote a simple program — a TSO command to display your own user ID. My program finds the user ID in the PSCBUSER field, by following full-word address pointers in control blocks, starting from the CVT (Communication Vector Table), which is the "backbone" of pointers to most locations on an MVS or OS/390 system. Figure 1 shows the actual simple code, and Figures 2 through 6 show LOOK screens that illustrate the storage at each control block location as we follow the chain to the location of the PSCB and the data we want. Using LOOK, you can now see what you're programming, so you're no longer working in the dark.

LOOK has many useful commands. You can enter a virtual address, and LOOK will take you there if the address is accessible. LOOK always has a current address pointer. Its initial value is 00000000 or low storage, and this is formatted as PSA (Prefixed Save Area). Indirect addressing is done with an "I" or a "J." A carryover from the 24-bit only days, "I" does indirect addressing in 24-bit mode. "J"does indirect addressing in 31-bit mode. Indirect addressing means that if a full-word in virtual storage contains an address of somewhere else in virtual storage, then a "J+0"command on the current full-word will take you to the address that's in that full-word. This corresponds to a LOAD instruction in assembler language. Therefore, the "J" instruction helps us follow control block chains. The code in the assembler program of Figure 1 has a sequence of five LOAD instructions. The corresponding commands to LOOK, illustrated by Figures 2 through 6, are a sequence of five "J" instructions. Figure 7 shows the LOOK help screen. LOOK doesn't have an enormous number of commands, but the commands it does have allow for great flexibility to track locations in virtual machine storage.

One other noteworthy aspect of LOOK is its ability to recognize and format system control blocks and to run from one control block to another through indirect LINKs. LOOK contains very clever code that allows you to format any control block fields,according to their macro definitions. In other words, LOOK can format fields in the CVT through an assembly of the CVT macro. The TCB can be formatted by invoking an assembly of the IKJTCB macro. Assembling the IHAASCB macro and so forth formats the ASCB. By coding a few lines in the CBMACS source member, any control block that is described by a mapping macro can be formatted to the LOOK program.

If a control block is formatted on a LOOK screen, you can get to any address in a formatted field by using a LINK or "L" command. For example, look at Figure 2, which is a formatted mapping of the CVT. If you want to display the location of the TCBP field from the CVT, you can enter the command LTCBP (or Link to TCBP), and you'll display storage starting at the location specified in the address, which is 00000218. So, starting from the CVT, you can get to the TCBP field by either entering "J+0," or "LTCBP," or "00000218" as commands. They will all yield the same result. Since a picture is worth a thousand words, there should be a lot of content in what I've said this month. Whether you're writing code or checking up on how your system is working, diagnosing a system problem, or simply exploring, it pays to be able to see the storage that's in the system. Good luck and good LOOKing. See you next month. ts

NaSPA member Sam Golob is a senior systems programmer.