

Writing User Exits in the C Language

Mike McNamee

mikem@e3sciences.com

Agenda

- The history of Exits
- Times are changing
- Changes to the C Compiler
- So where do I start ?

The history of Exits

- Users exits were written in Assembler because
 - IBM provided access (Macros) to many system services through assembler only
 - Mapping of system areas (DSECTS) only provided in an assembler friendly format
 - 3GLs provided poor performance because of run-time library overhead
 - Re-entrant code was not easily generated using 3GLs
 - Processors were much less powerful and we need to minimize overhead
 - System Programmers knew Assembler !!!

Times are changing

- New entrants to the industry do not know assembler - many know C/C++
- Many organizations do not place a high importance on training
- Performance is less of an issue

Times are changing

- USS is no longer an optional feature of OS/390 :
 - OS/390 V2R5 Comms Manager
 - HTTP Server
 - LDAP
 - Java
 - NetView Event Automation Service
 - Firewall

Times are changing

- C is the System Programming Language of UNIX
- Some new OS/390 facilities are not accessible in Assembler :
 - ICSF Secure Sockets Layer (V2R7)

Changes to the C Compiler

- IBM provides System Programming C (SPC) as part of OS/390 C/C++ :
 - No run time required
 - Self contained Load Module
 - Limited functionality
 - DSECT Conversion utility for translating data areas to C structures

Changes to the C Compiler

- SPC Provides a programming environment suitable for exit coding :
 - Compiler generates re-entrant code automatically
 - Low overhead
 - C Language suited to Systems Programming Tasks e.g. pointers
 - Provides a mechanism for non-assembler programmers to write exits

Changes to the C Compiler

- The disadvantages of using SPC :
 - Limited access to system services :
 - No linkage to WTO, ENQ, DEQ etc...
 - C can be difficult to read to the uninitiated
 - Limited C function access without runtime
 - Most functions not available
 - Is available under C only - not C++ compiler
 - No compliance checking in compiler for SPC limitations

C Functions available

- `abs()`, `fabs()`
- `memchr()`, `memcmp()`, `memcpy()`, `memset()`,
`memset_s()`, `cs()`
- `strcat()`, `strchr()`, `strcmp()`, `strcpy()`, `strlen()`,
`strrchr()`
- `malloc()` function
- `calloc()` function
- `realloc()` function
- `free()` function

So where do I start ?

- OS/390 C++ Programming Guide
 - Chapter 34 Using the System Programming C Facilities
 - Chapter 35 Library Functions for System Programming C

So where do I start ?

- Migrating existing DSECTs into C structures

– Execute HLASM with ADATA option

```
//PRMIKE1 JOB 5555,'COMPILE',MSGCLASS=X,NOTIFY=&SYSUID
//* -----
//ASSEM EXEC PGM=ASMA90,
// PARM='ADATA'
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR
// DD DSN=SYS1.MODGEN,DISP=SHR
// DD DSN=IMS.MACLIB,DISP=SHR
// DD DSN=PRMIKE.MACRO.SOURCE,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(1700,(400,400))
//SYSUT2 DD UNIT=SYSDA,SPACE=(1700,(400,400))
//SYSUT3 DD UNIT=SYSDA,SPACE=(1700,(400,400))
//SYSPUNCH DD DSN=PRMIKE.OBJ(SYSP),DISP=SHR
//SYSLIN DD DSN=PRMIKE.OBJ(SYSLIN),DISP=SHR
//SYSADATA DD DSN=PRMIKE.CPP.SOURCE(DSECT),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
OSIQUEUE
```

So where do I start ?

- Migrating existing DSECTs into C structures :
 - Convert to C structure using C/370 DSECT Utility using HLASM output

```
//*-----  
//* DSECT UTILITY STEP  
//*-----  
//DSECT      EXEC PGM=CBC3DSCT  
//STEPLIB   DD  DSN=SYS1.SCEERUN,DISP=SHR  
//          DD  DSN=SYS1.SCBCOMP,DISP=SHR  
//SYSADATA  DD  DSN=PRMIKE.CPP.SOURCE(DSECT),DISP=SHR  
//EDCDSECT  DD  DSN=PRMIKE.C370.SOURCE(DSECTO),DISP=SHR  
//SYSPRINT  DD  SYSOUT=*  
//SYSOUT    DD  SYSOUT=*
```

So where do I start ?

- Sample exit fragment

```

#pragma environment(FUSEX00,nolib) /* Define Runtime /Entry */
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct exit_dsect { /* DSECT from utility */
    int          exit_queue_addr;
    int          exit_return_code;
    int          exit_work_addr;
};

int FUSEX00()
{
    int *plist;
    struct exit_dsect *exitp;
    struct queue_entry_dsect *queue;
    char dsname??(60??), newname??(60??), empty = '\0', *dsnptr;

    plist = ( int * ) __xregs(1); /* Get Register 1 */
    exitp = ( struct exit_dsect * ) *plist; /* Get First Plist*/
    .....
    exit(0);
}

```

So where do I start ?

- #pragma environment defines entry point :
 - #pragma environment(*entry*,nolib)
 - Instructs compiler to generate prolog at the entry point with no additional initialization routine
 - nolib Generates linkage for SPC with no run time
 - *entry* names the function in the source code that is to be used as the entry point (can be > 1)
- C function provided to access plist :
 - __xregs

So where do I start ?

- Access to Operating System Services
 - Only compiler generated facilities available
 - A call to an assembler stub is required :

```
#pragma linkage(fuswto,OS) / * Use OS Linkage */
.....

memcpy(wto.rout,__wto11,2);
    memcpy(wto.desc,__wto3,2);
....

memset(wto.text,' ',70);
memcpy(wto.text,
        "ATM001E Too few parameters specified");
fuswto(wtoptr);
```

So where do I start ?

- Sample Assembler Stub

```

MODULE FUSWTO, BASE=12, LOC=BELOW, AMODE=31, RMODE=ANY, X
      TEXT='FUSEX00 WTO processor'
*-----
*   Establish Addressability
*-----
      L      R3,0(R1)           Get input struct
      USING INPUT,R3
*-----
*   Initialize WTO Plist
*-----
      MVC    SAVWTO,=AL2(124)   Move length
      MVC    SAVMCS,=XL2'8000'  Move MCS
      MVC    SAVDESC,INDESC    Move descriptor code
      MVC    SAVROUT,INROUT     Move route code
      MVC    SAVTEXT,INTEXT     Move message
      WTO    MF=(E,SAVWTO)      Issue write to operator
      ST     R1,INDOMID         Save DOM id
      ENDMOD
*-----
*   Constants ....
*-----
WFUSWTO  DSECT
SAVWTO   DC    AL2(124)
SAVMCS   DC    XL2'8000'
SAVTEXT  DS    CL120
SAVDESC  DC    XL2'0200'
SAVROUT  DS    XL2'4020'
SAVWTOL  EQU   *-SAVWTO
LFUSWTO  EQU   *-WFUSWTO
INPUT    DSECT
INROUT   DS    CL2
INDESC   DS    CL2
INDOMID  DS    CL4
INTEXT   DS    CL100
      END

```

So where do I start ?

- Producing a Load Module :
 - Compile, Pre-Link and Bind

```
//PRMIKE2 JOB MSGLEVEL=(1,1),MSGCLASS=X,NOTIFY=PRMIKE
//* -----
//*
//* This Job compiles, prelinks and links a batch C/370 program
//*
//S1 JCLLIB ORDER=(PRMIKE.C370.SOURCE)
//S1 EXEC EDCCPLG,INFILE='PRMIKE.C370.SOURCE(FUSEX00)',
// OUTFILE='PRMIKE.LOAD(FUSEX00)',CPARM='NOSTART,NOMAR,SOURCE,NOSEQ,RENT',
// LPARM='AMODE=31,MAP,RENT,REUS'
//COMPILE.SYSLIB DD DSN=CEE.SCEEH.H,DISP=SHR
// DD DSN=CEE.SCEEH.SYS.H,DISP=SHR
//* DD DSN=PRMIKE.C370.INCLUDE,DISP=SHR
//LKED.SYSLIB DD DSN=CEE.SCEESPC,DISP=SHR
// DD DSN=CEE.SCEELKED,DISP=SHR
// DD DSN=PRMIKE.LOADLIB,DISP=SHR
//LKED.SYSLMOD DD DSN=PRMIKE.LOADLIB,DISP=SHR
//LKED.SYSIN DD *
        ENTRY FUSEX00
        NAME FUSEX00(R)
```

So where do I start ?

- Producing a Load Module :
 - C has an extra Pre-Link step to handle long function names as CSECTs and DLL references e.g. GoAndGetThatLittleBitOfData(.....
 - To use SPC library on Bind step
 - CEE.SCEESPC
 - Compile must have NOSTART option to suppress LE initialization

So where do I start ?

- I keep getting abends !!!!!!! :
 - Using LE or C function (0C1, 0C4, U2xxx) ??
 - Look for some other way to achieve result
 - Using a str function and I get 0C4 ??
 - C Strings are NULL (x00) terminated
 - Using correct entry point ???

So where do I start ?

- My program/exit cannot live in the SPC environment - it's too restrictive !
 - Consider using pre-initialized LE environment :
 - CEEPIPI macro creates an LE environment
 - Removes overhead of continuous creating/deleting LE Process

