IBM Education Assistance for z/OS V2R3

Element/Component: XL C/C++

Material prepared for IBM Education Assistance and was current as of July 2017

Agenda

- Trademarks
- Session Objectives
- Overview
- Usage & Invocation
- Interactions & Dependencies
- Migration & Coexistence Considerations
- Feedback
- Session Summary
- Appendix

Trademarks

- See url http://www.ibm.com/legal/copytrade.shtml for a list of trademarks.

Session Objectives

- Show the major new enhancements to the C and C++ compilers in the following areas:
  - Usability:
    - Metal C Function Descriptors
    - Hexadecimal offsets for structure listings
    - DSECT zero extent arrays
  - Performance:
    - Architecture default changing to ARCH(10)
  - Security:
    - Stack protection
  - Debugging:
    - Metal C debug data blocks
    - Saved option string reading utility
    - DWARF debugging information in object files
  -

Session Objectives

- Show the major new enhancements to the C and C++ compilers in the following areas:
  - Usability:
    - Metal C Function Descriptors
    - Hexadecimal offsets for structure listings
    - DSECT zero extent arrays
  - Performance:
    - Architecture default changing to ARCH(10)
  - Security:
    - Stack protection
  - Debugging:
    - Metal C debug data blocks
    - Saved option string reading utility
    - DWARF debugging information in object files
  -

Usability

- Metal C function descriptors
- Hexidecimal offsets for structure listings
- DSECT zero extent array

Overview: Metal C function descriptors

- Problem Statement / Need Addressed
  - Using functions that could have their own context requires manual bookkeeping
    - Ex. Globals and statics
  -
- Solution
  - Create new function pointers that can act on environments as well as calling a function
  -
- Benefit / Value
  - Making function descriptors similar to the non-Metal C LE DLL mechanism allows similar coding patterns and automatic environment based calling

Usage & Invocation

1) Declare function pointers with the `__fdptr` keyword

2) Assign values to the function address and environment fields

3) Call a function through the function pointer

- Ex.

```
typedef int (* __fdptr remote_fptr_t)(int, int);

remote_fptr_t myDLLInit(void);

int main(void) {
  remote_fptr_t myRemoteFunctionPointer; // (1)
  myRemoteFunctionPointer = myDLLInit(); // (2)
  return myRemoteFunctionPointer(5, 11); // (3)
}
```

Overview: Hexadecimal offsets in listing

- Problem Statement / Need Addressed
  - The structure offset listings show decimal base offsets for structure members
    - Assembler listings and other listings may use hexadecimal base for offsets
-
- Solution
  - Add a suboption to the structure map to list offsets in hexadecimal
  -
- Benefit / Value
  - Layout information can be better compared and analyzed

Usage & Invocation

- A new suboption for the AGGREGATE option:

```
     +-NOAGG-------------+
>>---+-AGG-+-----------+-+---><
           +-OFFSETDEC-+
           +-OFFSETHEX-+
```

-
- Example USS usage:

Invoking:

```
> xlc -qaggregate=offsethex mysource.c
```

Results in:

```
...
===========================================================================================
| Aggregate map for: struct S1                           Total size: 56 bytes             |
|=========================================================================================|
|   Offset (Hex)    |      Length       | Member Name                                     |
|    Bytes(Bits)    |    Bytes(Bits)    |                                                 |
|===================|===================|=================================================|
|       0           |       1           |  m1                                             |
|       1           |       1(6)        |  m2                                             |
|       2(6)        |       1(2)        |  ***PADDING***                                  |
|       4           |       4           |  m3                                             |
...
```

Overview: DSECT zero extent array

- Problem Statement / Need Addressed
  - Structures generated by the DSECT utility are not always the same size as what was in the original DSECT
- Solution
  - Create trailing zero extent arrays to give the same size as the original DSECT
  -
- Benefit / Value
  - Creates C structures/unions that align closer to the original assembler DSECT

Usage & Invocation

- New option: NOLEGACY | LEGACY
  - NOLEGACY is the default and will generate the trailing zero extent array when needed
  - LEGACY will not generate the trailing zero extent array
- Example: The following DSECT gives the C structure shown

```
TEST           DSECT
FIELD1         DS      AL1
FIELD2         DS      AL2
FIELD3         DS      AL3
DSECTEND       DS      0D
LEN            EQU     *-TEST
               END
```

```
struct test {
  unsigned char  field1;
  unsigned short field2;
  unsigned int   field3 : 24;
  unsigned char  _filler1[2];
  __extension__ double dsectend[0];
};
```

Performance

- New minimum hardware support level

Overview: Architecture Level Set

-
- Problem Statement / Need Addressed
  - z/OS has moved up to a new minimum architecture
    - Code generated by the compiler should exploit that minimum hardware level
  -
- Solution
  - Change the default ARCH level to the new minimum hardware level
    - ARCH is now 10 by default, corresponding to zEC12
  -
- Benefit / Value
  - By default, code generated by the compiler will exploit at least the minimum hardware level that z/OS supports
    - Can change the default by explicit specification of ARCH or by use of the TARGET option to target a previous z/OS release

Usage & Invocation

- The ARCH level is set by default unless explicitly overridden
  - By explicit specification
  - By targeting an earlier release using the TARGET option

Migration & Coexistence Considerations

- To generate code for lower hardware levels on other systems, explicitly specify a lower ARCH level

Security

- Stack protection

Overview: Stack Protection

- Problem Statement / Need Addressed
  - Function return addresses are often used as an attack vector by overwriting them through a buffer overflow
  - Need a way to stop or detect overwriting of the return address
  -
- Solution
  - Protect buffers that are susceptible to overflow and do not return from functions that detect overwriting
  -
- Benefit / Value
  - Fails fast whenever there is stack corruption detected
  - Avoids an attack vector into applications

Usage & Invocation

- A new option, STACKPROTECT, and an associated INFO suboption have been added to protect buffers and warn of unprotected buffers respectively
  - NOSTACKPROTECT | STACKPROTECT(ALL|SIZE(<N>)) where N is the number of bytes, N>0
  - INFO(STP | NOSTP)

USS invocation command:

```
> xlc -qstackprotect=all -qinfo=stp mysource.c
```

There will be an LE ABEND (U4088-96) if stack corruption through buffer overflow is detected.

Interactions & Dependencies

- Software Dependencies
  - This feature will be added to V2R2 as well alongside the related Language Environment support
- Exploiters
  - Language Environment

Migration & Coexistence Considerations

- None for V2R3
  - Corresponding Language Environment APAR PI73324 needed for V2R2

Debugging

- Metal C debug data blocks
- Saved option string utility
- DWARF debugging information in object files

Overview: Metal C Debug Data Blocks

- Problem Statement / Need Addressed
  - Metal C generated objects and assembly and the associated debugging information may not be in sync if different debug files are used with different objects
  -
- Solution
  - Provide information linking the assembly or objects with the debugging data
    - Put the debugging side file name in the assembly
    - Provide a signature to ensure matched compilation time
  -
- Benefit / Value
  - Debugging information and the object and assembly files stay in sync
  - Helps catch out of date file errors earlier

Usage & Invocation

- Created by default under Metal C with DWARF debugging compilations
- Debug information block signature will be present in the debug information block: 0x'00C300C300D502vv' (vv = version)
  - Followed by the timestamp signature and source and debug file names
- The CDAHLASM or `as` utility will need write permission to the assembly file

USS invocation command:

```
> xlc -qmetal -S -qdebug=format=dwarf mysource.c
```

Overview: Saved Option String utility

- Problem Statement / Need Addressed
  - Knowledge of what options were used for generation of an executable are hard to learn after the fact
  - The need to know what options are in use by our users
- Solution
  - Provide a utility that allows emitting options encoded in the PPA blocks for feedback to the compiler team or for a user's own use
- Benefit / Value
  - Determining which options were in use to help diagnosing problems
  - Helping the compiler team focus our efforts into what our users are actually using

Usage & Invocation

- Example usage:

```
> /bin/sosinfo myexecutable
```

- The executable can be a USS path, a fully qualified dataset member name, a module name, an external link, etc.
- The procedure in Batch mode is CCNPSOS in CEE.SCEEPROC
- The module name is CCNESOS
- Example output:

```
ppa2_flt_ieee = NOIEEE
ppa2_service = NOSERVICE
ppa2_xpl_stargs = NOSTOREARGS
ppa2_charset = NOASCII
...
sos_arch = ARCH(8)
sos_tune = TUNE(8)
sos_csect = CSECT
sos_version_info = 9
...
```

Interactions & Dependencies

- Exploiters
  - Any of you!

Migration & Coexistence Considerations

- This utility is present from V2R1 and upwards after installation of the PTF's.

Overview: DWARF debug information in objects

- Problem Statement / Need Addressed
  - DWARF Debugging information is separate from the executable
    - Can get out of sync with each other or go missing
  - Cannot add it in general load sections due to increased memory footprint

- Solution
  - Put the debug data into the executable in an area that is not loaded at runtime
    - Have the debug data available upon request to be loaded if needed

- Benefit / Value
  - Lower starting memory footprint compared to the ISD format
  - Debug data and executable code stay together

Usage & Invocation

- A new suboption to DEBUG is now available
  - DEBUG(FILE) now becomes DEBUG(FILE | NOFILE)
    - NOFILE puts the debug information into the executable
    - Requires GOFF
    - DWARF format only

USS invocation command:

```
> xlc -qdebug=nofile mysource.c
```

Interactions & Dependencies

- Exploiters
  - dbx
  - dwarfdump

Feedback

- Always looking for feedback:
  - Contact: zosccpp@ca.ibm.com

Session Summary

- Showed the major new enhancements to the C and C++ compilers in the following areas:
  - Usability:
    - Metal C Function Descriptors
    - Hexadecimal offsets for structure listings
    - DSECT zero extent arrays
  - Performance:
    - Architecture default changing to ARCH(10)
  - Security:
    - Stack protection
  - Debugging:
    - Metal C debug data blocks
    - Saved option string reading utility
    - DWARF debugging information in object files

Appendix

- z/OS XL C/C++ Messages (GC14-7305-01)
- z/OS XL C/C++ Compiler and Runtime Migration Guide for the Application Programmer (GC14-7306-01)
- z/OS XL C/C++ User’s Guide (SC14-7307-01)
- z/OS XL C/C++ Language Reference (SC14-7308-01)
- Standard C++ Library Reference (SC14-7309-00)
- Common Debug Architecture User's Guide (SC14-7310-00)
- Common Debug Architecture Library Reference (SC14-7311-01)
- DWARF/ELF Extension Library Reference (SC14-7312-01)
- z/OS Metal C Programming Guide and Reference (SC14-7313-01)
- z/OS XL C/C++ Runtime Library Reference (SC14-7314-01)
- z/OS XL C/C++ Programming Guide (SC14-7315-01)
-
- z/OS Internet Library: http://www.ibm.com/systems/z/os/zos/bkserv/
- C/C++ Cafe - Community & Forum: http://www.ibm.com/rational/community/cpp