z/TPF EE V1.1
z/TPFDF V1.1
TPF Toolkit for WebSphere® Studio V3
TPF Operations Server V1.2

IBM Software Group

TPF Users Group Spring 2007

z/TPF GCC Compiler Update

Name: Peter Lemieszewski
Venue: Open Source Subcommittee

AIM Enterprise Platform Software
IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Agenda

- Definition & use of __attribute__
- Visibility support:
  - what is it?
  - What is it used for?
  - how to use it?
  - Migration support & z/TPF usage.
- More information
- z/OS compiler & z/TPF migration

Definition & Use of __attribute__

- __attribute__ assigns characteristics to:
  - Function declarations
  - Variable declarations
  - Type declarations, i.e. structure, union, enumerated types
- Syntax: __attribute__ ((attribute list))
  - Must appear before comma, equal sign, semicolon that terminates declaration
- Examples
  - void fatal () __attribute__ ((noreturn));
    - Function declaration fatal() will not return. Ever. Ever.
  - short array[3] __attribute__ ((aligned));
    - Align array on a machine dependent boundary (in z/TPF: word boundary)
  - struct my_packed_struct { …} __attribute__ ((__packed__));
    - Pack the structure “my_packed_struct”

Visibility support: what is it?

- Describes if & how a function is seen within & between modules.
  - **((visibility ("hidden")))**
    - No other module can directly call this function
    - Can be accessed indirectly via function pointers
  - **((visibility (“internal")))**
    - This function can never be called from another module
  - **((visibility (“protected")))**
    - This function is visible from another module
    - But all references within the module will be bound to local function only.
    - Useful when several functions have the same name.
  - **((visibility (“default")))**
    - This function is visible within the module and from any other module
- Visibility can be used as both an attribute and a compiler option.
- Visibility support included in gcc 4.0 & gcc 4.1

Visibility support: What is it used for?

- Visibility support refines the Global Offset Table (GOT) in shared objects.
  - References that do not have to be public can remain hidden

  In other words,
- Visibility Support can be used to eliminate the need for export files.
  - If the shared object is made up exclusively of C/C++ programs.

Visibility support: how to use it

- For a shared object's .mak file, specify:
  - -fvisibility=hidden
  - -fvisibility-inlines-hidden
- No functions are visible outside the shared object!
- Add __attribute__ (((visibility("default"))) to functions that are callable from other shared objects

Visibility support: Migration & z/TPF

- To migrate from z/OS compiler & TPF4.1:
  - transmogrify "`_Export`" to "`__attribute__((visibility("default")))`"
    - You can use Toolkit single source rule OTRKYWDc
  - If using "`NOEXPORTALL`" functionality, in z/TPF add to .mak files:
    - -fvisibility=hidden
    - -fvisibility-inlines-hidden
    - APP_EXPORT := VISIBLE
- Remember to apply z/TPF APAR PJ31477:
  - Build tool changes will now support visibility
    - If you want to use it
  - Can also be used as a model to for source file changes:
    - Obsolete .exp files
    - Use of "`__attribute__((visibility("default")))`"

For More Information

- GCC has a Wiki – information on all things GCC
  - http://gcc.gnu.org/wiki/HomePage
  - http://gcc.gnu.org/wiki/Visibility
- For more information on __attribute__
  - http://gcc.gnu.org/onlinedocs/gcc-4.0.0/gcc/Attribute-Syntax.html#Attribute-Syntax
  - http://gcc.gnu.org/onlinedocs/gcc-4.0.0/gcc/Function-Attributes.html#Function-Attributes
  - http://gcc.gnu.org/onlinedocs/gcc-4.0.0/gcc/Variable-Attributes.html#Variable-Attributes
  - http://gcc.gnu.org/onlinedocs/gcc-4.0.0/gcc/Type-Attributes.html#Type-Attributes
- "info gcc" on Linux.

z/OS compiler & z/TPF migration

- Before moving onto Linux:
- Two sets of changes will comprise majority of migration changes:
  - Constructs that syntactically changed or are no longer supported in z/TPF
    - Single Source APARs address these changes
    - IBM Toolkit's Single Source Scans can help manage the necessary changes
  - Constructs that are not ANSI standard (ISO/IEC 9899/1999 Standard)
    - z/OS V1R6.0 C/C++ Compiler or later can help manage the necessary changes
- Using z/OS compiler to manage ANSI compliance:
  - Move source code to a Unix System Services (USS) environment
  - Compile source code using compiler options:
    - LANGLVL(ANSI) : -W "0,langlvl(ansi),upconv"
    - ANSIALIAS: -Wc,ANSIALIAS
  - Note: Default options for z/OS compiler differ on USS & TSO.

z/OS V1R7.0 XL C/C++ User's Guide

- **ANSIALIAS | NOANSIALIAS**
- The cc compiler invocation command for a regular compile in the z/OS UNIX System Services environment uses NOANSIALIAS as the default option.
- The ANSIALIAS option indicates to the compiler that the code strictly follows the type-based aliasing rules in the ISO C and C++ standards, and can therefore be compiled with higher performance optimization of the generated code. When type-based aliasing is used during optimization, the optimizer assumes that pointers can only be used to access objects of the same type. Type-based aliasing improves optimization in the following ways:
- It provides precise knowledge of the data types to which pointers can and cannot point.
- It allows more loads to memory to be moved up and stores to memory moved down past each other, which allows the delays that normally occur in the original written sequence of statements to be overlapped with other tasks. These re-arrangements in the sequence of execution increase parallelism, which is desirable for optimization.
- It allows the removal of some loads and stores that otherwise might be needed in case those values were accessed by unknown pointers.
- It allows more identical calculations to be recognized (″commoning″).
- It allows more calculations that do not depend on values modified in a loop to be moved out of the loop (″code motion″).
- It allows better optimization of parameter usage in inlined functions.

- **LANGLVL(ANSI)**
- Use it if you are compiling new or ported code that is ISO C/C++ compliant. It indicates language constructs that are defined by ISO. Some non-ANSI stub routines will exist even if you specify LANGLVL(ANSI), for compatibility with previous releases. The macro __ANSI__ is defined as 1 for C only. It ensures that the compilation conforms to the ISO C and C++ standards.
- The UPCONV option causes the z/OS XL C compiler – only - to follow unsignedness preserving rules when doing z/OS XL C type conversions; that is, when widening all integral types (char, short, int, long). Use this option when compiling older z/OS XL C programs that depend on the *K&R C* conversion rules.

Questions?

PUT Levels & Lab Tested Compiler Releases

PUT LEVEL	COMPILER	LINUX PLATFORM SYSTEM LEVEL
PUT 0	GCC 3.4	Red Hat Enterprise Linux 3 -or- SUSE Linux Enterprise Server 8
PUT 1	GCC 3.4	Red Hat Enterprise Linux 3 -or- SUSE Linux Enterprise Server 8
PUT 2	GCC 4.0* or GCC 3.4	Red Hat Enterprise Linux 3 -or- SUSE Linux Enterprise Server 9
PUT 3	GCC 4.1	Red Hat Enterprise Linux 4 -or- SUSE Linux Enterprise Server 9

* Denotes compiler used to build OCO shared objects that were shipped on PUT

Trademarks

IBM, z/OS, System z, and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.
Other company, product, or service names may be trademarks or service marks of others.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Notes
Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

This presentation and the claims outlined in it were reviewed for compliance with US law. Adaptations of these claims for use in other geographies must be reviewed by the local country counsel for compliance with local laws.