TPF Users Group
Grapevine, Texas

IBM Software Group

C/C++ single source APARs

Languages Subcommittee

Sarat Vemuri

October 2004

AIM Core and Enterprise Solutions

IBM z/Transaction Processing Facility Enterprise Edition 1.1.0

Single Source APARs

- **Objective of "Single Source"**
  - Enable the same application source to be built for TPF 4.1 or z/TPF without any conditional code

- **Single Source APARs introduce changes into TPF 4.1 that are required for z/TPF**
- **Apply single source apars to your TPF 4.1 system**
  - Single source apars do not require any changes to applications - they do not break any existing interfaces
  - Single source apars enable you to change applications to make them compatible with z/TPF. The changes do not have to made at one time
- **Assess code to identify where changes need to be made**
  - For most cases scans will identify where changes can be made
  - Not all single source apars will require changes to your applications. The impact for some of the changes may be minimal to none.

TPF Single Source APARs

- **PJ29575 -- Add PTR32 type definitions**
- **PJ29630 -- Add the time_t32 and size_t32 definitions**
- **PJ29593 -- Add wrappers for header file changes and for the tpf directory**
- **PJ29576 -- Provide single-source packed decimal support**
- **PJ29692 -- Add the CPROC and CALLC macros**
- **PJ29640 -- Add the PRLGC, EPLGC, CSTKC and PBASC macros**
- **PJ29849 -- Add support for floating point migration (HFP and BFP)**
- **PJ29980 -- Provide API for conversion between native and HFP or BFP**
- **PJ29937 -- Add gettimeofday() to sys/time.h from sysgtime.h**
- **PJ29957 -- add time zone (TZ) environment variable**

APAR PJ29575 -- PTR32 Type Definitions and PTR32ATT Macro

- **Added 3 new typedefs that represent data (not function) pointers that are 32-bit addresses**

__ptr32_t	32-bit void pointer
__chptr32_t	32-bit char pointer
__uiptr32_t	32-bit unsigned int pointers

- **Added macro definition PTR32ATT**
  - to assist in the declaration of other pointers types of 32-bit
  - Example :typedef struct S * PTR32ATT __structSptr32_t;
- The new typedefs and the macro are provided in types.h header file
- **Why?**
  - In z/TPF, all pointers are 64-bit
  - For C/C++ structures that have equivalent Assembler DSECTs, changing the pointer field size will require assembler code change
  - If the pointer can still be 32-bit, it helps to declare the field to be a 32-bit pointer of appropriate type.

APAR PJ29575 -- continued

- **What to look for:**
  - Look at C/C++ structures that have equivalent assembler DSECT.
  - Also consider C/C++ structures that map data in a database/file.
  - Examine the pointer fields in those structures.
    - The following data will still be in 32-bit range in z/TPF
      - heap
      - stack
      - ECB private area
    - Note: Pointers to literal data and static data are not 32-bit.
- These pointers that are still valid 32-bit pointers could be changed to pointer types of 32-bit.
- Example:

```
__ptr32_t ce1cr0; /* This is a void pointer
                     of 32-bit */
```

APAR PJ29630 -- time_t32 and size_t32 definitions

- Provides typedef definitions for 32-bit data types
  - *time_t32, size_t32*, and *ssize_t32*
- The definition(s) of *time_t32*, *size_t32*, or *ssize_t32* are in whatever headers contain the definition(s) for *time_t*, *size_t*, or *ssize_t*
- **Why?**
  - In z/TPF, *time_t*, *size_t*, or *ssize_t* will be 64-bit data types
  - For C/C++ structures that have equivalent Assembler DSECTs, changing the field size will require assembler code change
  - If the field can still be 32-bit, it helps to declare the field to be a 32-bit field of appropriate type.
- **What to look for:**
  - Look at C/C++ structures that have equivalent assembler DSECT.
  - Also consider C/C++ structures that map data in a database/file.
- Fields of *time_t*, *size_t*, or *ssize_t* types may be changed to corresponding 32-bit types.
- **WARNING:** Do not pass a pointer to 32-bit data types
  - When required, assign data to a 64-bit data type and use its address

APAR PJ29593 -- Wrappers for header file name changes and tpf directory

- **Separates the TPF-unique header files from standard API header files in the hierarchical file system (HFS)**
  - Creates a new directory called ***tpf***
    - This is a subdirectory of the base *include* directory
    - It contains wrapper header files for TPF-unique headers
    - In the tpf directory, the dollar sign ($) in any header name is replaced with an underscore (_)
- For example:

  *.../include/**tpf**/tpfapi.h*

  *.../include/**tpf**/c_eb0eb.h*
- **All TPF-unique header files still exist with old names on TPF 4.1 but not on z/TPF**
  - Applying this APAR does not break existing application programs, even if the header files are in a PDS
    - The new ACP.CHDR.TPF.... data set should be appended after the old TPF header file data sets
  - This APAR enables customers to change code for single source one segment at a time.

APAR PJ29593 (continued)

- **Why?**
  - Created a separate directory to eliminate any future filename conflicts
  - Removed $ from file names to make name compatible with gcc compiler and linux rules
- **To make your application single source, you will need to make application source changes.**
  - Change all TPF-unique header file #includes by adding *tpf/* prefix
  - If the original header file name had a $, use the new name with underscore
  - For example,

    change *#include <c**$**eb0eb.h>*

    to *#include <**tpf/**c_eb0eb.h>*

NOTE: Header names are **case-sensitive** for z/TPF. This was not true for TPF 4.1

- If you have a #include header file name in uppercase, you may need to change to the actual header file name which may not be uppercase.

APAR PJ29593 (continued)

- **Tools to aid changes**
  - The following AS-IS tools will be available for downloading from the TPF web site when ready. See README files that come with the tools.
    - **add_tpf_hdr.sh** This tool adds tpf/ to the front of any TPF-unique header file referenced in the file.
    - **convert_hdr.sh** and **convert_src.sh** These tools change $ to underscore and changes #include "..." to #include<...>

APAR PJ29593 (continued)

- compare-and-swap *cs()* and compare-double-and-swap *cds() f*unctions provided in the *tpf/cmpswp.h* header on z/TPF
- **Why?**
  - On TPF 4.1, cs() and cds() are z/OS compiler built-in functions
  - On z/TPF, cs() and cds() were not gcc built-in functions
  - We provided them as TPF functions
- **What**
  - Current application code needs to be changed only to make it single source
  - In application programs that use *cs()* or *cds()*, add the following *#include* to your programs:
    - #include <tpf/cmpswp.h>

APAR PJ29576 -- Provide Single Source Packed Decimal Support

- **gcc compiler does not support the decimal data type**
  - The *decimal* data type is an extension of the zOS compiler
- **This APAR provides a compiler-independent API to support packed decimal data**
  - For C application:
    - *decNumber* library that contains the utilities that are needed to do decimal arithmetic in C programs
  - For C++ application:
    - *pDecimal*< , > in *pDecimal.hpp* header is a C++ class for the decNumber library
    - This class simplifies decimal arithmetic in the C++ environment and provides utilities for working with numbers in the familiar packed decimal format
- Code updates are simpler when using the pDecimal class.

APAR PJ29576 -- continued

```
#include <stdlib.h>
#include <stdio.h>
#include <decimal.h>

main(void)
{
  decimal(15,5)  op_1 = 245680.98786d;
  decimal(15,5)  op_2 =   5680.87675d;
  decimal(15,5)  result;

  result = op_1 - op_2;
  if (result == 240000.11111d)
    { printf(" Successful ! \n"); }

  printf(" The result is %15.2D(15,5) \n",result);
}
```

```
#include <stdio.h>                // for printf
#include <decNumMac.h>

int main(int argc, char *argv[]) {
  decNumber op_1, op_2, mtwo, result;    // working numbers
  char string[DECNUMDIGITS+14];
  decContextDefault

  decNumberFromString(&op_1, "245680.98786")
  decNumberFromString(&op_2,   "5680.87675")
  decNumberFromString(&mtwo,           "-2")

  decNumberSubtract(&result, &op_1, &op_2)    // result= op_1 -
op_2
  decNumberRescale(&result, &result, &mtwo);  // Call Rescale to
change
                              // the number of digits after the decimal point.
  decNumberToString(&result, string);

  if (!strcmp(string,"240000.11"))  /* result has only two digits of
acc. */
     printf(" Successful ! \n");

  printf(" The result is %15s \n",string); // 15 is the total field width
  return 0;
  } // main
```

APAR PJ29576 -- continued

```
#include <stdlib.h>
#include <stdio.h>
#include <decimal.h>

main(void)
{
  decimal(15,5)  op_1 = 245680.98786d;
  decimal(15,5)  op_2 =   5680.87675d;
  decimal(15,5)  result;

  result = op_1 - op_2;

  if (result == 240000.11111d)
    { printf(" Successful ! \n"); }

  printf(" The result is %15.2D(15,5) \n",result);
}
```

```
#include <iomanip>
#include <pDecimal.hpp>

using namespace std;

int main(int argc, char *argv[]) {
  pDecimal<15,2> result;
  pDecimal<15,5> op_1("245680.98786");
  pDecimal<15,5> op_2("5680.87675");

  result = op_1 - op_2;

  if (result == "240000.11")   // result has just two digits of
accuracy
     printf(" Successful ! \n");

  cout << " The result is " << setw(15) << result.toString() << endl;
  return 0;
  } // main
```

APAR PJ29640 -- PRLGC, EPLGC, CSTKC, and PBASC macros

- Four new assembler macros to code C library functions in assembler
  - PRLGC -- generates prolog code in library functions written in assembler, similar to the TMSPC macro
  - EPLGC -- generates epilog code in library functions written in assembler, similar to the TMSEC macro
  - CSTKC -- obtains or saves the address of the current C stack frame
  - PBASC -- gets or saves the address of the previous program base

APAR PJ29640 (continued)

- Existing functions written in assembler use the TMSPC and TMSEC macros as the prolog and epilog macros.
- Use the following macros to convert assembler-written C functions to single source.
- PRLGC
  - Use instead of TMSPC as the prologue.
- EPLGC
  - Use instead of TMSEC as the epilogue.
- CSTKC
  - Use instead of direct reference to CE3SPTR to save or restore the C stack frame pointer.
- PBASC
  - Use instead of direct reference to CSTKLBAS to save or restore the program base.
- For more information about these macros, see TPF General Macros

APAR PJ29640 (continued)

TPF 4.1 Library Function Example

BEFORE PJ29640

```
TMSPC ...
L       R13,CE3SPTR(R12,R9)
ST    R8,CSTKLBAS
L       R8,CE1SVP
CEBIC DBI,S
L       R13,CE3SPTR(R12,R9)
L       R8,CSTKLBAS
TMSEC RC=R1
```

AFTER PJ29640

```
PRLGC ...
 CSTKC GET=R13
 PBASC SAVER8=YES

 CEBIC DBI,S
 PBASC RESTORER8=YES

 EPLGC RC=R1
```

APAR PJ29692 -- Support for CPROC and CALLC Macros

- Parameter passing convention is different with z/OS compiler and gcc compiler.

- Two new assembler macros to encapsulate parameter passing from assembler to C programs
  - CPROC -- facilitates defining C program interface to assembler (similar to function prototypes in header files).
  - CALLC -- generates the code needed to call the C/C++ program from assembler. The macro sets up the parameters to the compiler convention.

- **Examine all C and C++ program calls from assembler.**
  - No code change is needed if the called program does not have any parameters.
  - For those that have parameters, use CALLC.
  - For each program that needs to be called, code a CPROC macro to describe the interface.

APAR PJ29980 -- Provide APIs for conversion between native and HFP or BFP

- **Floating point number representation in compiled programs differ between TPF 4.1 and z/TPF**
  - TPF 4.1 uses the zOS compiler to generate hexadecimal floating point (HFP) format for floating point numbers
  - z/TPF uses the gcc compiler. This compiler uses the binary floating point (BFP) format for floating point numbers, which is the IEEE standard format

- **Solution: Provide 4 floating point conversion functions for single source**
  - *tpf_fp_hton()* convert HFP to native format
  - tpf_fp_ntoh() convert native format to HFP
  - *tpf_fp_bton()* convert BFP to native format
  - tpf_fp_ntob() convert native format to BFP

  where "native" is HFP format for TPF 4.1 and BFP format for z/TPF

APAR PJ29980 (continued)

- TPF 4.1 applications that store floating point numbers on file as hexadecimal data (and not in character format such as 1e24) should be changed.
  - When data in file is HFP format, applications need to do the following:
    - After data is read into memory, issue the tpf_fp_hton() API for each floating point data item
    - Before writing the data to file, issue the tpf_fp_ntoh() API for each floating point data item
  - When data in file is BFP format, applications need to do the following:
    - After data is read into memory, issue the tpf_fp_bton() API for each floating point data item
    - Before writing the data to file, issue the tpf_fp_ntob() API for each floating point data item

- Note: These APIs do not handle long double
  - On z/OS compiler, long double is 16 bytes; on gcc compiler, it is 8 bytes
  - If you store data in file as long double, you must convert to double

APAR PJ29849 -- Floating Point Migration (HFP to BFP)

- At some point, you may want to convert your data on file to BFP format.
- Customer utility programs can use the following to convert HFP data to BFP and then write back to file.
  - *tpf_fp_htob()* convert a hexadecimal floating point (HFP) number to binary floating point (BFP) format

Note: this utility allows conversion from long double to double. It does not allow conversion to long double.

  - The following function is used by the 4 single source conversion APIs implemented by PJ29880
    - *tpf_fp_btoh()* convert a BFP number to HFP format

- **NOTE:** This is not a single-source APAR, but part of the solution for migration to z/TPF.

APAR PJ29937 -- Add gettimeofday() to sys/time.h from sysgtime.h

- For TPF 4.1, gettimeofday() is a TPF function
  - Prototype is in sysgtime.h header file
- For z/TPF, gettimeofday() now is a standard function
  - Prototype is in sys/time.h header file
- Scan for include of <sysgtime.h>.
  - Replace it with include of sys/time.h

APAR PJ29957 -- time zone (TZ) environment variable

- **Added support for environment variable "TZ".**
  - Allows to change local time zone information
- **In TPF4.1, locale category, TOD, is used to change the local time zone information.**
  - TOD category is an IBM extension to locale
  - z/TPF does not support the locale TOD category in locale
- **Scan for setlocale() function calls in applications.**
  - If the setlocale() is used to change time zone information
    - Replace it with setenv() to set "TZ" environment variable.
  - Example : setenv("TZ", "EST+5EDT,M10.5.0/2,M4.1.0/2")
  - To request local to GMT time difference be taken from the system is still supported. Code the local time difference to be >=24hrs.
  - Example : setenv("TZ", "EST+25EDT,M10.5.0/2,M4.1.0/2")

Legal

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

Other company, product, or service names may be trademarks or service marks of others.