Using GCC to Build CMS Applications

Dave Jones
V/Soft Software
Houston, TX

Agenda –

- Motivation
- The Package
- Installing the package
- Using the package
- Linux ABI for s390
- Calling C routines
- CMS Specific Libraries
- Sample code

Motivation

- Why do this? Because..
  - There's a lot of interesting software out there that's written in C
  - GCC is a really good compiler, and it comes standard with Linux, why not use it in CMS as well?
  - IBM C/C++ compilers cost $$$
  - IBM C/C++ compilers generate code that requires LE (not so good in a CMS environment)
  - Real reason: I needed the blowfish encryption algorithm

The Package

- C390.tar
- Written by John Hartmann in Summer/Fall of 2011
  - Wanted to port a Rexx interpreter, jRexx, from Linux to CMS
  - Wanted no LE-dependencies
  - Consists of CMS text libraries, documentation, utilities and C include files, as well as source code, both C and s390 assembler
  - User's Guide included

The Package

- c390.tar contains:

  c390.makefile – to build the package

  vma.patch – small patch the the Linux vmarc utility by Leland Lucius

  cmsh/390 - CMS-specific routines

  cmsh/edwards - Paul Edward's open source C library PDClib

  cmsh/include – C include files for 390 and edward's routines

  gcc390/ – code to convert gas to HLASM

The Package

CMS TXTLIBs -

- CMS390 - Base routines to create a stack frame and invoke the program with the expected argc/argv parameter list. Interface to callable routines and other CMS gutsy stuff.
- FPLPOP - Interface to the code underpinning the popen() function and friends. This allows a simple interface to CMS Pipelines. Generate a text library if have just the object deck:

  TXTLIB GEN FPLPOP FPLPOP

- PDP - Edwards' public domain C library with several fixes and enhancements of JPH. This includes the standard I/O library.
- GLIBC - A few routines JPH has lifted from the gnu C library. Notably the tree search functions.

Installing the package

On zLinux –

- Upload the c390.tar file to a convenient directory, usualy where you keep your source code, I use /home/dave
- Untar it
- Go to the gcc390 directory and issue the make command to compile the two utilities in that directory. Then move the two executable files to a directory in you path.
  - Makefile will likely need to be modified to meet your needs

Installing the package

On z/VM –

- Move the
  - CMS390 TXTLIB
  - GLIBC TXTLIB
  - LIBJ TXTLIB
  - PDP TXTLIB

  to CMS (as binary, F 80, of course)
- Move the *.macro files from gcc390/macros to CMS
- Generate GCC390 MACLIB, needed by HLASM

Using the Package

Compiling C source code -

To generate a gas assembler code file, you must use the following GCC options:

- -S
- -nostdinc
- -I
- -fexec-charset=IBM-1047
- -Wno-format
- -m31
- -march=g5
- -fno-use-linker-plugin

Using the Package

Compiling C source code, cont -

Some options that are useful:

- -fverbose-asm -g
- -save-temps
- -D__ZVM__ -D__CMS__ -U__gnu_linux__
- -Wall -Werror

Sample gcc invocation:

```
gcc -Wno-format -fexec-charset=IBM-1047 -Wno-format -Werror
-D__ZVM__ -D__CMS__ -U__gnu_linux__   -I$HOME/cmsh/include
-m31 -march=g5  -nostdinc  -fverbose-asm -save-temps  -S file.c
```

Using the Package

Converting gas to HLASM -

Use the two utilities provided, gas2asm and asmxpand

- gas2asm < file.s | asmxpand > file.assemble

Assemble -

Ether use the HLASM for Linux or other cross assembler on zLinux -or-

Move the assemble file to CMS and use HLASM there

- In ether case, you will need to use the macros provided or the assembly will fail

Using the Package

The Link step -

Always done on CMS, regardless of where the assembly step took place

Requires the following TXTLIBs be available

- CMS390
- FPLPOP
- PDP
- GLIBC

Using the Package

The Link step -

To link an application containing a C `main { }` function:

```
GLOBAL TXTLIB CMS390
LOAD RUN390 (CLEAR RLDSAVE NOINV
    NOUNDEF DUP
GLOBAL TXTLIB CMS390 FPLPOP PDP GLIBC
    LIBJ
INCLUDE main ( RLDSAVE NOINV UNDEF NODUP
GENMOD main (FROM RUN390
```

Using the Package

RUN390 code does:

- Allocates the run time stack (128KB default size)
- Sets afp bit in control register 0 so floating point operations can be done
- Allocates stdin, stdout, and stderr files with redirection
- Converts argument strings from CMS format (eplist) to C standard format (argc argv)
- Invokes the C main function
- Cleans up after the main function returns

Using the Package

The Link step -

To link an application that does not contain a C `main { }` function:

```
GLOBAL TXTLIB CMS390 FPLPOP PDP GLIBC
    LIBJ

LOAD main sub1 sub2 sub3  ( RLDSAVE

GENMOD main
```

Note: in this case, the “main” routine must set up the needed C environment like run390 does -

- Allocate a stack
- Set afp bit in cr 0 if needed

Linux ABI for s390

gcc generates code that follows the Linux ABI specification, defined here

http://refspecs.linuxfoundation.org/ELF/zSeries/lzsabi0_s390.html

Linux ABI is different than CMS, must use ABI to call C routines, pass arguments, manage stack space, etc.

Not hard, just different

Linux ABI for s390

Register usage

- Register 15 contains the stack pointer. The stack grows downwards.
- Register 14 contains the return address for a function call. The return address is not preserved.
- Registers 6-15 are saved/restored as needed across function calls. The called routine saves registers and restores them.

Linux ABI for s390

Register usage (cont)

- Registers 2-5 are used for the first four parameters, depending on size.
- The function result is in register 2.
- Registers 0-5 are used scratch registers. They are neither saved nor restored.

Calling C routines

To call a non-main C function:

- Allocate a stack of, say, 128KB
- Set reg 15 to point to top of stack – 96 bytes....this creates the first frame on the stack that is used by the first C routine called
- Set AFP bit in CR 0 if needed
- Allocate stdin, stdout and stderr files if needed....this can be done by a simple C routine
- Set needed parameters in Reg 2, Reg 3,...Reg 5
- When called C routine returns, free stack memory

CMS Specific Libraries

Functions specific to the CMS environment

- __adtlkp — Return an active disk table entry
- __svc204 — Call a CMS service
- cmssubcr — Enrol subcommand callback routine
- cmssubdl — Retract subcommand callback routine
- oscall — Call function expecting OS linkage
- tocsl — Interface to CMS callable service

Sample code

Rexx exec to generate an assembler file:

```
#!/usr/local/bin/rexx

/* simple Rexx exec to do a 390 gcc compile on C code */

/* for the s/390 environment....*/

trace Off

parse arg fn . '(' user_opts

options = '-Wno-format -fexec-charset=IBM-1047 -Wno-format
   -Werror '

options = options '-D__ZVM__ -D__CMS__ -U__gnu_linux__
   -I$HOME/cmsh/include  '

options = options '-m31 -march=g5  -nostdinc  -fverbose-asm
   -save-temps  -S '

options = options user_opts
```

Sample code

Rexx exec to generate an assembler file (cont);

```
'gcc' options fn'.c'
say 'rc=' rc
if rc = 0  then do
  'gas2asm <' fn'.s | asmxpand >' upper(fn)'.ASSEMBLE'
  if rc = 0 then do
      'vmur punch -t -r -u dave' upper(fn)'.ASSEMBLE'
   end
  end

exit
```

Sample code

Hello.c main program

```
/*                                                                   */
/*                                John Hartmann 16 Jun 2011 16:17:04 */
/*******************************************************************/
/* Change activity:                                                  */
/*16 Jun 2011  New module.                                           */
/*******************************************************************/

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <string.h>

#include <time.h>

#include <sys/time.h>
```

Sample code

Hello.c main program (cont)

```
#include <cmsbase.h>


/* Forward declarations:                                          */

/* End of forward declarations.                                   */


int

main(int argc, char ** argv)

{

   int i;

   FILE * f;
```

Sample code

Hello.c main program (cont)

```
char bfr[256];

char * s;

size_t lns=0;

size_t bytes=0;

struct timeval tv;

struct tm * tm;

int rv;


for (i=0; argc>i; i++)

   __sayf("%3i: %.50s", i, argv[i]);
```

Sample code

Hello.c main program (cont)

```
    gettimeofday(&tv, NULL);
  tm=localtime(&tv.tv_sec);
  __says(bfr, strftime(bfr, sizeof(bfr), "Time %T date %Y-%m-%d %w %a %Z", tm));
  __sayf("ctime %s tv %ld usec %d", ctime(&tv.tv_sec), tv.tv_sec%86400, tv.tv_usec);


  __sayf("Sizeof int %ld long %ld size %ld",
     sizeof(int), sizeof(long), sizeof(lns));
 f=fopen("LOAD.MAP", "r");
  if (!f)
  {
     printf("coudnt open");
     exit (2);
  }
```

Sample code

Hello.c main program (cont)

```
 __sayf("file %p %m fd %d", f, fileno(f));
for (;;)
{
   s = fgets(bfr, sizeof(bfr), f);
   if (!s) break;
   lns++, bytes+=strlen(bfr);
}
printf("%zd lines %zd bytes\n", lns, bytes);
fclose(f);
rv = fprintf(stderr, "This on std error\n");
fprintf(stderr, "Got rv %d\n", rv);
printf("This on stdout\n");
```

Sample code

Hello.c main program (cont)

```
  f=popen("literal hello there|cons|count lines|cons", "w");
if (!f) printf("popen fail %m\n");
else
{
   printf("pipe fd %d open\n", fileno(f));
   i = fputs("here goes\ntwo lines\n", f);
   if (ferror(f)) __sayf("fd %d %m", fileno(f));
   printf("closing pipe fd %d\n", fileno(f));
   pclose(f);
}
```

Sample code

Hello.c main program (cont)

```
    printf("hello");

  printf(" %m");

  return system("id");

}
```

Sample code

Hello.c main program output:

```
Ready; T=0.03/0.04 08:44:43

hello

  0: hello

Time  8:44:46 date 2013-06-15 6 Sat CST

ctime Sat Jun 15 08:44:46 2013C tv 53086 usec 318195

Sizeof int 4 long 4 size 4

file 03F3EE88 No error fd 3

223 lines 22423 bytes

This on std error

Got rv 18

This on stdout

hello there
```

Sample code

Hello.c main program output:

```
pipe fd 3 open

here goes

two lines

closing pipe fd 3

3

DAVE     AT ZPDTVM61 VIA RSCS     06/15/13 08:44:46 CST      SATURDAY

hello No error

Ready; T=0.08/0.11 08:44:46
```

Sample code

CALLC assembler example

```
            TITLE 'A ROUTINE FOR CALLING GCC C code'
CALLC    CSECT ,
CALLC    AMODE 31
CALLC    RMODE ANY
@MAINENT DS    0H
         USING *,@15
         B     @PROLOG
         DC    AL1(16)
         DC    C'CALLC     13.165'
         DROP  @15
@PROLOG  STM   @14,@12,12(@13)
         LR    @12,@15
```

Note:
This example does not set the AFP bit in CR 0, nor does it open the three standard in/out/err files. See the code in RUN390 ASSMEBLE and GO390 C files for examples of how to perform that.

Also, please don't use this as an example of good assembler coding techniques.....

Sample code

CALLC assembler example (cont.)

```
@PSTART  EQU   CALLC
         USING @PSTART,@12
         ST    @13,@SA00001+4
         LA    @05,@SA00001
         ST    @05,8(,@13)
         LR    @13,@05
*      Key = '0123456789ABCDEFF0E1D2C3B4A59687'x;
         MVC   KEY(16),@CB00053
*      IV = 'FEDCBA9876543210'x;
         MVC   IV(8),@CB00056
*      Clear_text =
*       '37363534333231204E6F7720697320746852074696D6520666F722000'x;
```

Sample code

CALLC assembler example (cont.)

```
         MVC   CLEAR_TEXT(32),@CB00058
*      Text_len = 32;
         LA    @08,32
         ST    @08,TEXT_LEN
*  /*  ?DMSMSG TEXT('clear text: &1')
*         SUB(HEX4,Clear_text);*/
*    /* allocate storage for the rexx program descriptor array */
*      Generate Refs(Stack_size);
@GS00021 DS    0H
         CMSSTOR OBTAIN,DWORDS=Stack_size,BNDRY=PAGE
@GE00021 DS    0H
*      Stack_bottom = Reg1;
```

Sample code

CALLC assembler example (cont.)

```
          ST    REG1,STACK_BOTTOM
*       stack_top = Stack_bottom + Stack_size * 8;
          LR    @09_STACK_TOP,REG1
          L     @05,STACK_SIZE
          SLA   @05,3
          ALR   @09_STACK_TOP,@05
*       reg15 = Stack_top - 96;
          LR    REG15,@09_STACK_TOP
          LA    @08,96
          SLR   REG15,@08
*       reg2 = 0;
          SLR   REG2,REG2
```

Sample code

CALLC assembler example (cont.)

```
*       reg3 = 0;
          SLR   REG3,REG3
*       /* build the parm list, gcc style */
*       /* make the context;  void BLFMKCT (int * handle)  */
*         reg2 = Addr(CTX);
          LA    REG2,CTX
*       Generate Refs(reg14);
@GS00028 DS    0H
         L   reg14,=v(BLFMKCT)
         basr reg14,reg14
@GE00028 DS    0H
*       /* set the encryption key */
```

Sample code

CALLC assembler example (cont.)

```
*      /* void BLF128K(int  * handle, unsigned char *key) */
*        reg2 = Addr(CTX);
         LA    REG2,CTX
*        Reg3 = addr(Key);
         LA    REG3,KEY
*      Generate Refs(Stack_size,Stack_bottom);
@GS00031 DS    0H
         L   reg14,=v(BLF128K)
         basr reg14,reg14
@GE00031 DS    0H
*      /* set the IV             */
*      /* void BLFIV(int *handle, unsigned char *key      */
```

Sample code

CALLC assembler example (cont.)

```
*         reg2 = Addr(CTX);
          LA    REG2,CTX
*         Reg3 = addr(IV);
          LA    REG3,IV
*       Generate Refs(reg14);
@GS00034 DS    0H
          L   reg14,=v(BLFIV)
          basr reg14,reg14
@GE00034 DS    0H
*       /* encrypt the text       */
*       /* void BLFENBK(int *handle, unsigned char *blk, int *len) */
*         reg2 = Addr(CTX);
```

Sample code

CALLC assembler example (cont.)

```
         LA    REG2,CTX
*        Reg3 = addr(Clear_text);
         LA    REG3,CLEAR_TEXT
*        Reg4 = Addr(text_len);
         LA    REG4,TEXT_LEN
*      Generate Refs(reg14);
@GS00038 DS    0H
         L   reg14,=v(BLFENBK)
         basr reg14,reg14
@GE00038 DS    0H
*      /* free the context data structure */
*
```

Sample code

CALLC assembler example (cont.)

```
*         reg2 = Addr(CTX);
          LA    REG2,CTX
*       Generate Refs(reg14);
*
@GS00040 DS    0H
         L   reg14,=v(BLFFREE)
         basr reg14,reg14
@GE00040 DS    0H
*    /* free the storage for the rexx descriptor array */
*       Generate Refs(Stack_size,Stack_bottom);
*
@GS00041 DS    0H
```

Sample code

CALLC assembler example (cont.)

```
        CMSSTOR RELEASE,DWORDS=Stack_size,ADDR=Stack_bottom
@GE00041 DS    0H
*   /* we're outta here.... */
*      Return code(0);
         SLR   @15,@15
         L     @13,4(,@13)
         L     @14,12(,@13)
         LM    @00,@12,20(@13)
         BR    @14
*   End;
@DATA    DS    0H
         DS    0F
```

Sample code

CALLC assembler example (cont.)

```
@SA00001 DS    18F
         DS    0F
         LTORG
         DS    0D
STACK_SIZE DC  F'16384'
STACK_BOTTOM DS A
RETURN_CODE DS F
CTX      DS    F
TEXT_LEN DS    F
@CB00058 DC    XL29'37363534333231204E6F7720697320746865207469 6D6520666*
               F722000'
         DC    3X'0'
```

Sample code

CALLC assembler example (cont.)

```
@CB00053 DC    XL16'0123456789ABCDEFF0E1D2C3B4A59687'

@CB00056 DC    XL8'FEDCBA9876543210'

KEY      DS    CL16

IV       DS    CL8

CLEAR_TEXT DS  CL32

@DYNSIZE EQU   0

@00      EQU   0

@01      EQU   1

@02      EQU   2

@03      EQU   3

@04      EQU   4

@05      EQU   5
```

Sample code

CALLC assembler example (cont.)

```
@06      EQU   6

@07      EQU   7

@08      EQU   8

@09      EQU   9

@10      EQU   10

@11      EQU   11

@12      EQU   12

@13      EQU   13

@14      EQU   14

@15      EQU   15

@09_STACK_TOP EQU @09

REG0     EQU   @00
```

Sample code

CALLC assembler example (cont.)

```
REG1     EQU   @01

REG2     EQU   @02

REG3     EQU   @03

REG4     EQU   @04

REG5     EQU   @05

REG6     EQU   @06

REG7     EQU   @07

REG8     EQU   @08

REG9     EQU   @09

REG10    EQU   @10

REG11    EQU   @11

REG12    EQU   @12
```

Sample code

CALLC assembler example (cont.)

```
REG13    EQU   @13

REG14    EQU   @14

REG15    EQU   @15

         DS    0D

@ENDDATA EQU   *

@MODLEN  EQU   @ENDDATA-CALLC

         END   CALLC
```

To do:

Things to try –

- Using gcc on Intel Linux, cross-compiling
- Use HLASM on Linux instead of on CMS
- Use the free Tachyon Legacy Assembler
- Build 64 bit applications for use on zCMS

Who am I?

I'm –

Dave Jones

V/Soft Software

Real expertise in virtual technologies

www.vsoft-software.com

How to reach me –

dave@vsoft-software.com

281.578.7544

Thanks!

Thank you for coming today.....