EXPLOITING VM/CMS REXX WITH WATERLOO C

ERIC GIGUERE AND DOUG MULHOLLAND
Waterloo

Exploiting VM/CMS REXX Facilities

with Waterloo C

Doug Mulholland

University of Waterloo

Internet: dwm@csg.uwaterloo.ca
Voice: (519) 888-4676

Overview

Run-time Library Facilities

- REXX function and variable access
- XEDIT command and file buffer access
- CMS SUBCOM support

Debugger Support

- C debugger macros

C Compiler (Preprocessor) Extension

- compile-time REXX interface

C Calls to REXX

ANSI Standard Interface

- system() – C program calls <fname> EXEC

  int system(char *cmdstr);

  Example:

  system("myexec command-line text");

  - parse arg
  - exit return_code

Waterloo C Additions: <csubcom.h>, <rexxdef.h>

- execft() – call a REXX macro with an initial default address environment:

  int execft(char *cmdstr, char *ftype);

  Example:

  execft("PROFILE command-line text", "MYAPPL");

- execrexx() – call a REXX program in memory

  int execrexx(char *cmdstr, char *ftype,
  size_t stmtcount, char **rexxstrs);

Example: execrexx()

```
#include <csubcom.h>
#include <stdio.h>
#include <string.h>

static char *rxpgm[] =
  {
    "/* INMEM REXX */",
    "say 'INMEM REXX: address() =' address()",
    "parse arg cmdline",
    "say ' cmdline = \"'cmdline'\"'",
    "parse source srcline",
    "say ' srcline = \"'srcline'\"'",
    "exit length( cmdline )"
  };

#define CMDLINE "Typical command line"

int main()
  {
    int stmtnum, retval;
    char *rxcmdline;

    /* print the REXX program, then execute it... */
    printf( "char *rxpgm[]...\n" );
    for( stmtnum = 1; stmtnum <= 7; ++stmtnum )
        printf( " %d: %s\n", stmtnum, rxpgm[ stmtnum - 1 ] );

    printf( "CMDLINE = \"%s\"\n strlen( CMDLINE ) = %d\n",
        CMDLINE, strlen( CMDLINE ) );
    rxcmdline = "INMEM " CMDLINE;
    retval = execrexx( rxcmdline, "REXX", 7, rxpgm );
    printf( "execrexx( rxcmdline, \"REXX\", 7, rxpgm ) = %d\n",
        retval );
  }
```

Example: execrexx() ...continued

Output:

```
char *rxpgm[]...
 1: /* INMEM REXX */
 2: say 'INMEM REXX: address() =' address()
 3: parse arg cmdline
 4: say ' cmdline = "'cmdline'"'
 5: parse source srcline
 6: say ' srcline = "'srcline'"'
 7: exit length( cmdline )
CMDLINE = "Typical command line"
 strlen( CMDLINE ) = 20
INMEM REXX: address() = REXX
 cmdline = "TYPICAL COMMAND LINE"
 srcline = "CMS COMMAND INMEM REXX * !INMEM REXX"
execrexx( rxcmdline, "REXX", 7, rxpgm ) = 20
```

REXX Variable Access

Stem Variable Access: <stdio.h>, <file.h>

```
fopen( "_REXX.MYVAR.", "r+" )
open( "_REXX.MYVAR.", O_WRONLY|O_TRUNC|O_CREAT )
```

- compatible with EXECIO (STEM
- MYVAR.0 contains the number of lines (n)
- MYVAR.n contains the "file" data
- all the usual I/O functions: fprintf(), fscanf(), read(), write(), ...
- supports direct access: fseek(), ftell()
- command line redirection of standard I/O

Single Variable Access: <rexxdef.h>

- C library functions to set, get and drop a REXX variable's value

```
int rexxset( char *var, char *value );
int rexxfetch( char *var, char *bufptr, size_t buflen );
int rexxdrop( char *var );
```

Program Stack Access

I/O using <stdio.h>, <file.h>

```
fopen( "_STACK.FIFO", "w" )
open( "_STK", O_WRONLY|O_TRUNC|O_CREAT )
```

- FIFO, LIFO (the default)
- compatible with REXX push, queue, parse pull
- all the usual I/O functions: fprintf(), fscanf(), read(), write(), ...
- direct access using fseek(), ftell() is diagnosed as an error
- command line redirection of standard I/O

REXX Calls to C

"Traditional" Calls to C Programs

```
/* */
'MYPGM Typical C program command-line'
say 'program return code:' rc
```

- various forms of argc/argv command line processing:
  - UNTOKENIZED: argc is 1, argv[0] is the entire command line
  - TOKENS: usual CMS tokenization (8 character, upper case tokens)
  - EXTEND: tokenization (without truncation)
  - C_STRINGS (UNIX compatible): quoted strings preserved as one token, trigraphs and '\' processed as for C strings
- integer return value from main() assigned to REXX rc

Function Calls to C Programs

```
/* */
rexx_string = mypgm( "parm 1", "parm 2" );
say 'mypgm returned:' rexx_string
```

- C program can dynamically detect environment and (optionally) exit with a string return value

```
int isrexxfn( void );
void rexxit( char *str );
```

XEDIT Access

XEDIT Subcommands

xedit(char *str);

Example

```
xedit( "MSG Bye, bye file!" );
xedit( "ALL\\:1 DEL *" );
```

XEDIT File Buffer Access

- <depsets.h>

  int setxedit(int enable_xedit);

  0 disables access, 1 enables access

- <stdio.h>, fopen(), ...

  when a file in the XEDIT file buffer ring is opened, I/O is performed through the XEDIT SUBCOM

- no CMS minidisk/SFS access required

Example Usage

- compiler can compile from an XEDIT file buffer, write errors to a second file buffer, and display a message in the editor message area
- when run from XEDIT, DIFF can compare the disk version of a file with the current file buffer contents
- CALC (a calculator program) can append its output to an editor file buffer

CMS SUBCOM Support

<csubcom.h>

- C program defines a subcommand handler function: subcomset()
- program calls REXX (or just another program component), usually with execft
- handler is called with (argc/argv) command line parameters
- run-time environment recovered by SUBCOM handling interface (signal handling, open files, etc.)
- return code from handler returned to REXX
- handler function removed from SUBCOM list with subcomclr()
- can be used to transfer data between programs

Example: subcomset(), subcomclr(), execft()

```
#include <csubcom.h>
#include <stdio.h>
#include <string.h>

static int schandler( int argc, char **argv );

int main()
  {
    csubcom_descr *csdptr;
    char *cmdline, *ftype;
    int retval;

    csdptr = subcomset( schandler, "SUBC1" );
    cmdline = "C2REXX2C";
    ftype = "SUBCSET";
    printf( "main: calling execft( \"%s\", \"%s\" )\n",
          cmdline, ftype );
    retval = execft( cmdline, ftype );
    printf( "main: retval = %d\n", retval );
    subcomclr( csdptr );
  }

static int schandler( int argc, char **argv )
  {
    int i;

    printf( "schandler() : argc = %d\n", argc );
    for( i = 0; i < argc; ++i )
          printf( " argv[ %d ] = \"%s\"\n", i, argv[ i ] );
    return( argc );
  }
```

Example: subcomset(), subcomclr(), execft() ...continued

```
/* C2REXX2C SUBCSET -- demonstrate C calling REXX calling C */

address SUBC1
say 'C2REXX2C SUBCSET: calling SUBC1...'
'Alphanumeric, mixed-case subcommand line'
say 'C2REXX2C: rc =' rc
exit 1
```

Output:

```
main: calling execft( "C2REXX2C", "SUBCSET" )
C2REXX2C SUBCSET: calling SUBC1...
schandler() : argc = 5
  argv[ 0 ] = "SUBC1"
  argv[ 1 ] = "Alphanumeric,"
  argv[ 2 ] = "mixed-case"
  argv[ 3 ] = "subcommand"
  argv[ 4 ] = "line"
C2REXX2C: rc = 5
main: retval = 1
```

C Debugger Support for REXX

CDEBUG Provides:

- a CMS SUBCOM entry point for REXX macros to call
- PROFILE CDEBUG, (NO)PROFILE <fname> command-line option
- EXTRACT subcommand for accessing debugger internal data
- OPTION OUTPUT <file-name> lets command output be written to a file, including _REXX.<stemmed-var>

Example:

```
/* CALC CDEBUG */
parse arg cmdline
address command 'MAKEBUF'
bufnum = rc
address cms 'CALC' cmdline '>_stk.fifo'
calcrc = rc
if calcrc = 0 then do queued()
      parse pull calcout
      'MSG' calcout
   end
address command 'DROPBUF' bufnum
if calcrc <> 0 then
   __"EMSG Return code '"calcrc"' from 'CALC'."
exit 0
```

Compile-time REXX Support

An Experimental Facility!

- compiler recognizes a builtin C preprocessor symbol to call REXX ("external") functions:

  __EXT__(FUNCNAME)(text of REXX argument string)

- FUNCNAME EXEC is called as a REXX function, string return value is inserted into C source stream
- compiler provides a SUBCOM entry point for accessing internal (symbol table) data

Compile-time REXX Support ...continued

Example:

```
#include <stdio.h>

int main()
   {
      printf( "hello, world\n" );
      __EXT__(SYSCMD)( CP Q TIME )
      return( 0 );
   }


/* SYSCMD EXEC - execute a system command */
parse arg cmdline
say 'SYSCMD:' cmdline
address cms cmdline
exit ""
```

Output:

```
Ready;
cw crexx
Waterloo C (Version 3.3B IBM 370)
SYSCMD. CP Q TIME
TIME IS 13:49:50 EDT TUESDAY 05/11/93
CONNECT= 00:32:24 VIRTCPU= 000:39.60 TOTCPU= 000:53.52
File 'crexx c a1': 8 lines, included 130, no errors
Ready;
```

- it's no worse than what the SQL preprocessor strips out, and certainly better than what the C preprocessor can do!

Compile-time REXX Support – Applications

So What Can We Do With It?

- access compiler data: the compiler SUBCOM entry point supports "TYPEOF" and "SYMLOOK" directives (extract symbol table data)
- call MAKE to do other updating operations (enforce updatedness)
- insert source code into the program: the REXX function's return value is "compiled"

  Example:

```
__EXT__(SRC1)()          /* HELLO C */


/* SRC1 EXEC */
src =       '/* HELLO C -- A First Program. */'
src = src 'int main()'
src = src ' {'
src = src '   printf( "hello, world\n" );'
src = src '   return( 0 );'
src = src ' }'
exit src
```

Compile-time REXX Support – Applications

- evaluate non-constant expressions

  Example:

```
#define SIN _ _EXT_ _(CWSIN)

const double sin_of_5 = SIN(5);


/* CWSIN EXEC */
parse arg sinarg
address cms 'CALC sin(' sinarg ') >_REXX.CALCOUT.'
parse var calcout.1 skip1 '=' cwsinval '(' skip2
exit cwsinval
```

- retrieve C source code from somewhere (e.g., network, database, ...), "pre-preprocess" it, then compile it

Summary

Programs and Applications

- in UNIX environments, pipes and shell programs provide the "glue" to combine programs into "applications"
- for CMS, and now MVS and OS/2, REXX lets application programmers (and even end-users) combine programs and customize applications
  - transfer program control between programs within an application
  - transfer data between programs within an application