Computer Optics Vol. 2, No. 2, pp. 199–206, 1990
Printed in Great Britain

0955-355X/90 $10.00 + 0.00
Copyright © 1990 Pergamon Press plc

INTERFACING OF INTERLANGUAGE MODULES IN RSX-11M AND RT-11

L. I. BRUSILOVSKY and YU. A. MIKHAILOV

Abstract—The paper considers practical procedures for interfacing Pascal-2 and FORTRAN IV interlanguage modules in the RSX-11M and RT-11 operating systems; organizing processing operations for variable length files in the Pascal-2/RT-11 programming language; transfer of binary records in excess of 512 bytes from RSX-11M to RT-11; and addressing service functions of the monitor of the operating system from the programs written in Pascal-2.

1. INTRODUCTION

Computer hardware provides the key for building any integrated automation system, for example, the integrated automation system for computer synthesized optics [1]. Because most modern computers have a developed system support, the user can choose an operating system that suits best the role the computing resources play in the automated system. Flexibility of the software is achieved by using a common high-level language in dfferent operating systems. FORTRAN IV and Pascal-2 are examples of such languages for the 16-bit PDP-11 family widely used in integrated automation systems [2–4].

Pascal-2 compilers available in both operating systems (RSX-11M and RT-11) allow for efficient use of computer memory. The local object store is not made a part of the memory for the executable file, but rather is allocated when required. This is quite essential because of the limited memory space under the operating systems. It is worth noting that very many RSX-11M and RT-11 applications have been written in FORTRAN IV (e.g. the wide variety of scientific and engineering program libraries). Thus, the interfacing of the Pascal-2 and FORTRAN-IV modules is of great practical significance. In addition the PASMAC software package included in the Pascal-2 development systems of both operating systems (to facilitate interfacing to OSs written in an assembly language) does not permit automatic addressing of global variables, which might complicate software development and maintenance. It may be expedient therefore to use a pipe, or software interface, which is a part of both OSs. The pipe is essentially a system subroutine library which can be accessed using a conventional FORTRAN scheme (CALL statement).

Although the original language of the Pascal-2 compiler allows the invocation of external FORTRAN-IV modules, there are a number of constraints imposed on direct access to the NONPASCAL module call mechanism which is integrated into Pascal-2.

First, both compilers generate an environment set-up code to run programs written in the corresponding languages. Primarily, this concerns the control structures which manage input/output (IO) and initialize the stack (through register 6). The IO procedures of the languages are different; therefore IO operations using facilities built in a language must be executed within modules written in the same language as the main program. (Modules written in a language other than that of the main program can run physical IO operations by way of access to the operating system functions.)

Second, the compilers operate with different external module call mechanisms: Pascal-2 controls parameter transfer through the *stack* (the description in the external NONPASCAL procedure/function causes the compiler to generate, during invocation of the function, a code sequence similar to that generated by the FORTRAN-IV compiler, that is, load the address of the actual parameter address list into register 5). However even if the standard call sequence is used in interfacing Pascal and FORTRAN modules, the associated problems may be as follows:

(1) how to transfer an external module name as an actual parameter (the name of a Pascal-2 external module cannot be transferred directly to a FORTRAN module because of the features inherent in code generation if an external procedure name is sent as a parameter),
(2) how to access Pascal global variables and such in a FORTRAN module and, conversely, to access COMMON variables in Pascal-2 modules.

Binary files consisting of variable-length records which are more than 512 bytes in length cannot be transmitted directly between RSX-11M and RT-11 (because RSX-11M utility, called FLX, that transmits files between these OSs, handles records up to 512 bytes in length). Below we consider two simple Pascal-2 programs that easily circumvent these problems.

We shall now demonstrate practical methods of interfacing the interlanguage modules (Pascal-2 to FORTRAN-IV) in RSX-11M and RT-11, manipulation of variable-length files in Pascal-2/RT-11, and transmission of binary records in excess of 512 bytes between RSX-11M and RT-11.

2. PASCAL–FORTRAN INTERFACING

2.1. *Generation of external module load address*

A FORTRAN external module load address is generated by calling a special function embedded in the FORTRAN-IV programming systems of both OSs (e.g. IADDR in RT-11). It will be recalled that for the PDP-11 family, an integer in the range 0–65,535 is taken to be such an address. An external module address in a Pascal procedure can be generated, for example, by the following independently compilable auxiliary function (to relieve the compiler of matching the types of actual and formal parameters):

```
(* $ nomain *)
type
  ADDRESS : 0 . . 65535;
function ADDR(I : ADDRESS) * ADDRESS; external;
function ADDR; (* function body ADDR *)
begin
  ADDR:=I
end;
```

If we call the ADDR function as an actual parameter, the name of the external module (declared as EXTERNAL) must be specified, and the address has to be generated. By way of example, imagine that the name of the external module EXT (as an actual parameter) should be transferred from a Pascal program to a FORTRAN subprogram. The calling sequence will be as follows:

```
. . . . . . . . . .
type
  ADDRESS : 0 . . 65535;
. . . . . . . . . .
procedure EXT; external;
function ADDR(procedure F) : ADDRESS; external;
procedure FORT(I : ADDRESS); nonpascal;
. . . . . . . . . .
  FORT(ADDR(EXT));
. . . . . . . . . .
```

It is instructive to note that whatever language the EXT module is written in, it must be declared EXTERNAL rather than NONPASCAL. It is of no significance whether or not it has formal parameters and is a procedure (subroutine) or a function. It is essential that the parameter (address of the external module load point) is sent to the FORTRAN module by value.

Consider a Pascal-2 module which may be called from a FORTRAN-IV module. Once an external module has been called, the FORTRAN-IV compiler loads the address of the actual parameter address list into register 5. The low byte in the first word of the list contains the number of the actual parameters while the addresses of the parameters begin in the second word. Suppose that three parameters are transferred from a FORTRAN module to a Pascal-2 module: an integer, a real array and a real number. To cancel generation of the array length check code, the Pascal-2 module should be compiled with the keyword NOCHECK. The formal parameter list can be simulated as follows:

```
(* $ nomain *)
procedure EXT; external;
procedure EXT;
type
  VECTOR = array 1 . . 100 of real;
  (* parameter address list *)
  PARAMETER_LIST = record
    NP  : integer; (* number of parameter *)
    AI  : integer; (* 1st parameter address *)
    AV  : VECTOR; (* 2nd parameter address *)
    AR  : real; (* 3rd parameter address *)
    end;
var
  (* R5 address and contents *)
  R5 origin 177058 : PARAMETER_LIST;
..........
begin
  K := R5^.AI^; (* K = 1st actual parameter value *)
..........
```

If the calling sequence were PASCAL–FORTRAN–PASCAL, the global variables in the above module would be accessed, if necessary.

2.2. *Access to shared variables*

Consider the order of calling Pascal-2 global variables from FORTRAN-IV modules.

The Pascal-2 compiler loads global variables into the program section with the name GLOBAL (or with the name of the corresponding module provided the compile keyword/OWN is used) and with a 4-byte displacement from the origin of the section. Thus, to access such variables in the FORTRAN-IV module it is essential that the required variable be declared in the COMMON block with the name GLOBAL (or other appropriate name) and with a 4-byte displacement from the origin of the block:

```
var (* global variables *)
  A  : real;
  B  : array [1 . . 200] of boolean;
  C  : record
    AX  : integer;
    ST  : array [1 . . 100] of char
      end;
      SUBROUTINE F1(PARAMETERS, IF ANY)
      COMMON/GLOBAL/DUMMY,A,B,AX,ST
      REAL A
      BYTE B(200),ST(100)
      INTEGRAX
..........
```

In this example, DUMMY stands for a dummy variable which provides the 4-byte displacement from the origin of the COMMON block named GLOBAL. The method is not applicable where access to the COMMON variables in a Pascal-2 module is required since the Pascal-2 executive uses the first four bytes of PSECT allocated by global variables, consequently the information written by these variables would be lost. In this case, the method of access to the COMMON block variables may be as follows. A FORTRAN-IV subroutine should be written which could return the addresses (16-bit integers) of the COMMON load points (e.g. using the system function IADDR in RT-11). For a Pascal-2 program, access-type variables should be chosen to match these addresses and be of the base type corresponding to the COMMON block definition.

Assume that access to variables of the common blocks COM1, COM2 and COM3 in a Pascal-2 procedure, is required. The appropriate FORTRAN-IV subroutine may be as follows:

```
SUBROUTINE CADDAR(ADDR)
INTEGER ADDR(1)
COMMON/COM1/A(100),I(10)
COMMON/COM2/B(200),J(300),F
COMMON/COM3/Z
ADDR(1) = IADDR(A)      ! or ADDR(1) = IADDR(COM1)
ADDR(2) = IADDR(B)      ! or ADDR(2) = IADDR(COM2)
ADDR(3) = IADDR(Z)      ! or ADDR(3) = IADDR(COM3)
RETURN
END
```

---

and the Pascal-2 calling sequence may be:

```
type
  COMADDR = array [1 . . 3] of integer;
  CM1 = record
    A  : array [1 . . 100] of real;
    I  : array [1 . . 10] of integer;
    end;
  CM2 = record
    B  : array [1 . . 200] of real;
    I  : array [1 . . 300] of integer;
    end;
     CM3 = real;
   ACOM = CM1;
  ACOM2 = CM2;
  ACOM3 = CM3;
. . . . . . . . . .
var
  COM1  : ACOM1;
  COM2  : ACOM2;
  COM3  : ACOM3;
  ADDR  : COMADDR;
. . . . . . . . . .
procedure CADDR(var A : COMADDR); nonpascal;
. . . . . . . . . .
begin
. . . . . . . . . .
  CADDR(ADDR);
  COM1: = loophole(ACOM1,ADDR[1]);
  COM2: = loophole(ACOM2,ADDR[2]);
  COM3: = loophole(ACOM3,ADDR[3]);
. . . . . . . . . .
  K: = COM1^.I[L] + COM2^.[1];
. . . . . . . . . .
```

---

3. FILE MANIPULATION

3.1. *Pascal variable-length records*

All Pascal-2 records of nontextual files are regarded as having a fixed-length defined by the file type. Manipulation of Pascal variable-length records can be based on independent compilation of modules. A file containing variable-length records should be defined as follows:

```
type FBYTE = file of integer
```

while the input/output of variable-length records should be organized by means of two Pascal-2 procedures, WVL (output) and RVL (input):

```
(* $ nomain *)
type
  FBYTE = file of integer
  ABYTE = array [1 . . 10] of integer;
procedure WVL(var F:FBYTE; var X:ABYTE; K:integer); external;
procedure WVL;
var
  I : integer;
begin
  write(F,K)  (* write record length *)
  (* write structure *)
  for I: = 1 to (K + 1)div2 do write(F,X[I])
end;
procedure RVL(var F:FBYTE; var X:ABYTE; K:integer); external;
procedure RVL;
var
  I : integer
begin
  read(F,K);  (* read record length *)
  (* read structure *)
  for I: = 1 to (K + 1)div2 do read(F,X[I])
end;
```

The WVL (writing) procedure can be called as

```
type
  FBYTE = file of integer;
  VREC1 = record . . . end;
  VREC2 = record . . . end;
. . . . . . . . . .
var
  F1,F2 : FBYTE;
  X     : VREC1;
  Y     : VREC2;
  I     : integer;
. . . . . . . . . .
procedure WVL(var F:FBYTE; I,LEN:integer); external;
. . . . . . . . . .
procedure RVL(var F:FBYTE; I:integer; var LEN:integer); external;
. . . . . . . . . .
```

```
begin
..........
  WVL(F1, loophole(integer,ref(X)),size(VREC1));
..........
  RVL(F2, loophole(integer, ref(Y)),I));
..........
end.
```

This record structure is identical with that of variable-length records in RSX-11M.

3.2. *Processing of variable-length character strings*

Some global processing problems arise from the severe restriction of types characteristic in Pascal-2. The string processing package included in the Pascal-2 development system does not provide the necessary flexibility since all the strings must be of the same maximum length regardless of application. The problem can be easily tackled by the system object library SYSLIB which contains string processing subroutines and variable-length string mechanism (end of string is identified as the first occurrence of a null byte). String addresses (in the form of integers) rather than the strings themselves should be specified as actual parameters of the system subroutines. The addresses should be passed by value. By way of example, we present a function defining a string length:

```
..........
var S: packed array [1 . . 81] of char;
..........
function LEN(I : integer): integer; nonpascal;
..........
  K: = LEN(loophole(integer, ref(S)));
..........
```

3.3. *Transfer of variable-length records exceeding* 512 *bytes from RSX*-11*M to RT*-11

This transfer problem arises in the transfer of files prepared by the unformatted input statement of FORTRAN RSX-11M to RT-11, when the record length of a file exceeds 512 bytes. As a way out we recommend the following algorithm:

(i) Use PIP/RSX-11M routine to transform the file into a continuous form (by the keyword/CO incorporated in PIP).
(ii) Print out the file heading by means of DMP/RSX-11M routine and determine the first block address in the disk file (virtual block 0).

The following actions are carried out under RT-11.

(iii) Use the command line COPY/DEV/FIL DV1:/START:STARTADDRESS/END: ENDADDRESS DV2:FILNAM.EXT, to copy the file from the RSX-11M-format disk to the RT-11-format disk ENDADDRESS is made up of the address of block 0 in the disk file (determined by DMP) and the size of the transferred file.
(iv) If the file is to be read out of Pascal-2 modules, then no other procedures are required. This file can be read using the above-mentioned RVL. If the file is to be handled by the FORTRAN serial unformatted input statements, the two following routines should be executed. The first of them, written in Pascal-2, reads the file and creates an intermediate file. The second, written in FORTRAN IV, reads the intermediate file and generates unformatted records in FORTRAN IV/RT-11.

```
type
  F1: file of integer;
  F2: text;
  I,J,K: integer;
```

```
begin
  reset(F1, NAME); rewrite(F2, NAME);
  while not eof(F1) do begin
    (* record length *)
    read(F1,I); write (F2,I);
    for J: = 1 to (I + 1)div2 do begin
      read(F1,K)
      writeln(F2,K)
    end
  end
end.
```

```
      BYTE M(MAXIMUM LENGTH IN BYTES)
      INTEGER N(MAXIMUM LENGTH IN WORDS)
      EQUIVALENCE(M,N)
      OPEN(UNIT = 1, NAME = 'INTFILENAME'),DISPOSE = 'DELETE')
      OPEN(UNIT = 2, NAME = 'OUTFILENAME'),FORM = 'UNFORMATTED')
1     READ(1, *, END = 3) L ! LENGTH IN WORDS
      DO 2 I = 1, (L + 1)/2
2     READ(1, *) N(I)
      WRITE(2)(M(I), I = 1,L)
      GOTO 1
3     STOP
      END
```

---

4. ACCESS TO MONITOR FROM PASCAL PROGRAM

The monitor can be called directly from a Pascal-2 program by using the predetermined EMT procedure. A call of procedure in the program is equivalent to issuing the EMT instruction with a code equal to the EMT argument value. For example, a call to EMT (375B) is equivalent to the instruction EMT 375.

Note that prior to call EMT, the EMT argument block address must be loaded into the register R0. Assume that a Pascal-2 program has to pass its data to another job by the SDATW system call. The EMT argument list takes on the following format:

```
R0 ⇒  ___________
        25     0
      ___________
        not used
          BUF
          WCNT
           0
      ___________
```

where BUF is the address of the data transmission buffer, and WCNT is the buffer length in words.

A section of the Pascal-2 program may be as follows:

```
const   BUFLEN = 1000B;
type    BUFFER = array [1 . . BUFLEN] of integer;
        ARGBLOCK = record
          CODE: packed array [1 . . 2] of 0 . . 255
          DUMMY: integer;
          BUF: ^BUFFER;
          WCNT: integer;
          ENDA: integer;
        end;
```

```
  var     BUFF: BUFFER;
          EMTARG: ARGBLOCK;
          R0 origin 177700B : ^ARGBLOCK; (* for PDP-11/40 *)
begin
..........
          with EMTARG do begin
            CODE[1]: = 0;
            CODE[2]: = 25B;
            BUFF: = ref(BUFF);
            WCNT: = BUFLEN;
            ENDA: = 0;
          end;
          R0: = ref(EMTARG);
          EMT(375B);
```

REFERENCES

1. A. E. Bereznyi, L. I. Brusilovsky *et al.* Project of computer-aided design, manufacturing, and application of plane optical elements, Version 1. *Komputernaya Optika* No. 2, MTsNTI, Moscow (1987).
2. *PASCAL-2: Version 2.0 for RT-11*, S.L. (1981).
3. *PDP-11 PASCAL/RSX-11 Programmer's Guide.* Digital Equipment Corp., Maynard (1983).
4. *PDP-11 Fortran Language Reference Manual.* Digital Equipment Corp., Maynard (1979).