Concurrent Pascal User's Guide

Mattsson, Sven Erik

1979

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7167)/1-009/(1979)

CONCURRENT PASCAL USER'S GUIDE

SVEN ERIK MATTSSON

DEPARTMENT OF AUTOMATIC CONTROL
LUND INSTITUTE OF TECHNOLOGY
AUGUST 1979

Dokumentutgivare
04T0 Lund Institute of Technology
Handläggare Dept of Automatic Control
06T0 Sven Erik Mattsson
Författare
08T0 Sven Erik Mattsson

Dokumentnamn
REPORT
Utgivningsdatum
August 1979

Dokumentbeteckning
LUTFD2/(TFRT-7167)/1-009/(1979)
Ärendebeteckning
06T6

10T4

Dokumenttitel och undertitel
18T0 Concurrent Pascal User's Guide.

Referat (sammandrag)
26T0 This document together with the Concurrent Pascal Report define Concurrent Pascal for the LSI-11 computer.

Referat skrivet av
42T0 Author

Förslag till ytterligare nyckelord
44T0

Klassifikationssystem och -klass(er)
50T0

Indextermer (ange källa)
52T0

Omfång
96T0 96 pages

Övriga bibliografiska uppgifter
56T2

Språk
English

Sekretessuppgifter
60T0

ISSN
60T4

ISBN
60T6

Dokumentet kan erhållas från
62T0 Department of Automatic Control
Lund Institute of Technology
Box 725, S-220 07 Lund 7, Sweden

Mottagarens uppgifter
62T4

Pris
66T0

DOKUMENTDATABLAD enligt SIS 62 10 12

SIS-DB 1

Blankett LU 11:25 1976–07

INTRODUCTION

This document is intended to be read in conjunction with the Concurrent Pascal Report (Brinch Hansen, P.,'The Architecture of Concurrent Programs', Prentice-Hall Inc., Englewood Cliffs, New Jersey Ø7632, 1978). It describes the restrictions and extensions for the implementation on the LSI-11 computer. The specifications stated in this document are preliminary and may be subject to changes.

LANGUAGE RESTRICTIONS AND EXTENSIONS.

Character Set

The special characters are extended with

[ ] { }

The set letters is extended with lower case letters, but they will be interpreted as their upper case counterparts.

Basic Symbols

The special symbols are extended with

! [ ] { } /* */ (* *)

The word symbols are extended with AND.

Four forms of comment brackets are allowed: "...", {...}, (*...*) and /*...*/. The opening and closing comment bracket must have the same form.

The boolean operator & can be written AND. The boolean operator OR can be written as !.

The array brackets (. and .) can be written [and].

A string constant must be contained in one source line. If the string contains a quote mark (') then this quote mark must be doubled ('').

Types

A nonstandard enumeration type cannot consist of more than 128 constant identifiers.

Integer case labels must be in the range Ø..127.

The range of integers is -32768..32767. Real numbers have the range 1ØE-38..1ØE+38, with a precision of 7 digits.

A one-dimensional array of type boolean or char must contain an even number of elements.

A set may contain at most 64 elements.

A process component can only be declared within the initial process.

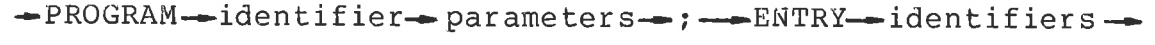
The standard procedure continue can only be called within a routine entry of a monitor.

SEQUENTIAL PROGRAMS

A process can execute a program written in Pascal and compiled with the Pascal compiler. In the sequel such a program will be referred to as a sequential program or a Pascal program.

Declarations in the concurrent program

The process type must include a declaration of the sequential program:



A program declaration consists of program identifier, a parameter list and a list of access rights.

There must be exactly one variable parameter and no constant parameter, and the variable parameter must be of the following type:

```
type seqprogspectype=record
                       filename: array[1..14] of char;
                       stacktop,
                       heaptop,
                       startaddress,
                       loadaddress: integer
                     end;
```

Note that the user must declare this type and that the compiler can only check that the parameter has the correct length.

The access rights of a program are specified by a list of identifiers of routine entries defined within the process in which the program is declared. The sequential program may call these routines during its execution. There must be at least two identifiers in the list.

The first one should be a procedure which inputs a single character from the terminal. It will be used implicitly by read and readln in the sequential program. The procedure

should have one variable parameter of type char.

The second one should be a procedure which outputs a single character to the terminal. It will be used implicitly by write and writeln, and also to print error messages from the sequential program. The procedure should have one variable parameter of type char. These two procedures must be defined by the user.

A sequential program must be loaded into the store before it can be called by the process. The loading of a sequential program is handled by the following standard procedure:

```
loadseq(seqprogspec,varspace)
```

Seqprogspec should be a variable of the type seqprogspectype (defined above) and varspace an integer. The filename component of seqprogspec should contain the filename of a relocatable version of the sequential program. Loadseq will give the rest of the components of seqprogspec appropriate values. The data space needed to execute the sequential program (in words) should be specified by the user in varspace.

Loadseq uses the RT-11 disk handler. In order to load sequential programs safely, all sequential programs should be loaded before any process is initialized. A sequential program is not reentrant, but it can be restarted without reloading. If two processes want to execute the same sequential program, two copies must be loaded.

The parameter used at a call of a sequential program should contain the values received at loading.

Declarations in the sequential program

The sequential program should follow the rules of (OMSI-) Pascal. Read and write can only be done from and to the terminal.

A sequential program may use the routines specified in the ENTRY list. All routines in this list except the first two must be declared in the sequential program. The first two are used implicitly (see above) and must not be declared in the sequential program. The rest of the routines in the list are called explicit entry routines.

Suppose that there are n explicit entry routines in the ENTRY list, then declare an enumeration type and an external procedure:

```
type entryroutines=(entry1, entry2,...,entryn);
procedure entprocedure(entryroutine:entryroutines);external;
```

Suppose that the i:th explicit entry routine is a procedure

then it should be declared in the following way:

```
procedure routinei({the same parameterlist as in the
                    declaration of the procedure in
                    the process});
 begin
   entprocedure(entryi)
 end;
```

Note that all parameters of entry routines must be variable parameters. Records with two simple successive elements of type boolean or char or records with a simple element of these types at the end are not allowed as parameters of an entry procedure.

It is natural but not necessary to give the Pascal procedure the same name as in the Concurrent Pascal process. The important issue is that entprocedure(entryi) will call the i:th explicit entry routine if entryi is the i:th element of the type entryroutines. Moreover the parameter lists must be compatible. Neither the Pascal nor the Concurrent Pascal compiler can check these two things.

Explicit entry routines which are functions turned out to be impossible to implement efficiently. Therefore they are not allowed.

Preparation of the sequential program

Create a relocatable version of the sequential program under RT-11 in the ordinary way. Compile the source code with the OMSI-Pascal compiler and assemble it with MACRO. Link the object file with LINK.In order to create a relocatable version the R-switch should be used. Two support libraries are needed. They are called PASLIB.OBJ and CPASUP.OBJ. In a normal case the command line to LINK should have the form

```
FILNAM=FILNAM,PASLIB,CPASUP/R
```

It is important that PASLIB is specified before CPASUP.

PROCESS ATTRIBUTES

The standard function

```
attribute(x)
```

returns an attribute of the calling process. The index and value of the attribute are universal enumerations. The attribute index x is of the following type:

```
type attrindex = (caller, prinumber)
```

The attribute function can be used as follows:

attribute(caller) The result is an integer that identifies the calling process. The system associates consecutive integers 1, 2, ... with processes during their initialization, starting with the initial process.

attribute(prinumber) The result is the current priority of the calling process as defined in the next section.

PROCESS SCHEDULING AND REAL-TIME CONTROL

A priority is associated with every process. It is described by a nonnegative integer. A smaller priority number corresponds to a higher priority. All scheduling is done according to the priority. If two processes have the same priority, the first-come-first-served rule applies. The initial process will start with priority zero. The other processes will start with priority one.

A process can change its priority during execution by means of the standard procedure:

```
setpri(x)
```

The priority of the calling process is set to x. If x<Ø a runtime error occurs. Changes of priorities are significant events and a process may be suspended if it lowers its priority.

The standard routines for real-time control are:

```
realtime
```

The standard function realtime returns an integer defining the real time in ticks mod 32768 after system initialization. The time between two ticks is 2Ø ms and 32768 ticks are 1Ø minutes and 55.36 seconds.

```
sleep(x)
```

The calling process is delayed until the time defined by the standard function is x. Only one process at a time can sleep. If a process calls sleep when another process is sleeping, or if x<Ø, a runtime error occurs. If the sleeping is done in a monitor this will delay other calls of the monitor.

awake

A process can wake up a sleeping process by calling awake. If no process is sleeping the call is ignored.

INPUT/OUTPUT

At present the only I/O device is the console terminal. Input/output is handled by the following standard procedure:

io(x,y,z)

Peripheral device z performs operation y on the variable x. The calling process is delayed until the operation is completed. X should be a variable parameter of type char. Y should be a variable parameter of type ioparam and z should be a constant parameter of type iodevice. Ioparam and iodevice should be declared as

```
type iodevice = (typedevice);
     ioparam = (input, output)
```

The character CR is input as LF and echoed as CR,LF. The character LF is output as CR,LF.

A concurrent program must ensure that the device is not used by more than one process at a time.

OTHER STANDARD FUNCTIONS

The standard function

gateopen(x)

applies to monitors. The result is a boolean value, true if the monitor is free, false otherwise. Note that the use of this function does not violate any of the strict access rules of Concurrent Pascal. The intention is that a high priority process may check the availability of a monitor before trying to enter it. If the monitor is not free, the process may choose some alternative action.

COMPILER

The compiler is named CPAS and runs under RT-11. The command line to the compiler follows the rules of RT-11 and has the following structure:

objectfile,listfile=sourcefile/switch1.../switchn

The objectfile, which contains the executable program, will have the default extension EMU, the listfile LST and the sourcefile will have CPA.

The following switches may be used:

/N The generated code will only identify line numbers of the program text at the beginning of routines. This reduces the code by about 25 percent, but makes error location more difficult.

/C The code will not make range checks of constant enumeration arguments.

/O If a listfile is specified, the emulator code will be output between the source lines in the listfile. If no listfile is specified, the emulator code will be output on the device TT.

/T Has the same effect as switch O with the addition that the intermediate code of all passes will be output. (This facility is used as a diagnostic aid to locate compiler errors.)

If an error is detected, no objectfile will be created.

EXECUTION

The execution of a program written in Concurrent Pascal is controlled by a program called KERNEL. KERNEL should be started as a normal RT-11 program. After the asterisk printed by KERNEL, the name of the file containing the compiled Concurrent Pascal program should be entered. The filespecification should follow the rules of RT-11 and the default extension is EMU.