Pascal-Plus

Another Language for Modular Multiprogramming

J. Welsh & D. W. Bustard

Basics

- 1. Model Operating Systems (PPP)
- 2. SIMONE
- ICL 1900 Series (Queen´s University of Belfast)

Syntax

- envelope

- process
- monitor
- condition

envelope

ist eine Blockstruktur und definiert:

- eine Datenstruktur
- Operationen
- eine Kontrollstruktur

Beispielkonstrukt

```
envelope histogram;
        const N = 10;
        type range = 0...N;
        var count : array [range] of integer;
           i: range;
        procedure *record (value: range);
           begin count [value] := count [value] + 1 end;
        begin
           for i := 0 to N do count [i] : = 0;
           ***;
           writeln (´value occurences´);
           for i : = 0 to N do writeln (i: 4, count [i]: 12)
        end;
```

1. count wird initialisiert
2. Ausführung des inneren Statement (***)
3. Statement, um count auszugeben wird ausgeführt

Das innere Statement führt Block B aus, in der die Instanz H deklariert ist:

instance *H: histogram;* oder
instance *H1, H2: histogram;*

H.record(j) oder
with *H* **do begin** *…; record(j); …; record(k);…* **end**

wie man sieht kann man statt der Punktnotation auch eine with-Schleife verwenden.

Hierarchie bzw. Ordnung

- Wenn mehrere envelope-Instanzen in einem Block deklariert sind, ist die Ordnung durch die Ordnung der Deklarationen bestimmt: Eine Instanz umschließt immer die nachkommende.

Wenn aber nur eine envelope-Instanz deklariert ist, wird die Definition und die Deklaration kombiniert zu einem Modul (envelope module):

```
envelope module H;
        const N = 10;
            :
            defined as before
            :
        end;
```

Jeder Bezeichner, der in einem envelope definiert ist, ist starred; nicht nur seine Prozeduren oder Funktionen.

Starred bedeutet, dass der Bezeichner keinen Wert, sondern eine Adresse beinhaltet oder erwartet.

Auf eine starred Variable hat man immer nur ein Leserecht, d.h. der Block, der eine envelope-Instanz schafft, darf starred-Variablen keinen Wert zuweisen!

Der reine Lesezugriff bewirkt, dass die „use“ und „import“-Deklarationen weitestgehend weggelassen werden können; dieses werden ja in anderen Programmiersprachen unbedingt benötigt.

- envelope-Konstrukt vergleichbar mit „class“ in Concurrent Pascal oder mit „module“ in Euclid
- „envelope module“ vergleichbar mit „module“ in Modula

- process
- monitor
- condition

Diese 3 Konstrukte werden für die parallele Programmierung unbedingt benötigt.

process

Ein Prozess ist definiert als ein Block mit einem Kopf, der wie der envelope-Kopf, Parameter enthält.
Instanzen werden genauso deklariert, wie die Instanzen eines envelope.

```
program producers and consumers;
        type itemtype = …;
        … module Buffer … {see later} …;
        process producer;
           var i: itemtype;
           begin
                repeat
                   produce item i;
                   Buffer.put (i)
                until switch off
           end;
        process consumer;
           var i: itemtype;
           begin
                repeat
                   Buffer.get (i);
                   consume item i
                until switch off
           end;
  instance
        P: array  [1…3] of producer;
        C: array  [1…4] of consumer;
  begin
        ***
  end.
```

Was passiert nun, wenn der Puffer voll ist, der Produzent aber weiter produzieren will ?

…die Einführung von Warteschleifen...

monitor

In einem Monitor werden mehrere Prozeduren und Variablen zusammengefasst. Die Idee ist nun, dass in jedem Monitor eines Programms sich gleichzeitig immer nur ein Prozess befinden darf. Falls ein weiterer Prozess in den Monitor eintreten will, muss er sich schlafen legen und warten bis der andere Prozess den Monitor verlassen hat.

Nun verwenden wir einen Monitor um sicherzustellen, dass immer nur ein Prozedur auf den Puffer zugreift, bzw. diesen verändert.

```
monitor module Buffer;
        var item: array [0...99] of itemtype;
           putcount, getcount: integer;
        instance notfull, notempty: condition;
        procedure *put (i: itemtype);
           begin
                if putcount-getcount = 100 then notfull.wait;
                item [putcount mod 100] : = i;
                putcount : = putcount + 1;
                notempty.signal
           end;
        procedure *get (var i: itemtype);
           begin
                if putcount-getcount = 0 then notempty.wait;
                i : = item [getcount mod 100];
                getcount : = getcount + 1;
                notfull.signal
           end;
begin putcount : = 0; getcount : = 0; *** end;
```

condition queues

Condition Warteschleifen enthalten die Operationen:

- wait
- signal
- (pwait)

- Simulation
- Real-Time