STUDIA UNIV BABEŞ-BOLYAI, MATHEMATICA, XXXVIII, 3, 1993

AN IMPLEMENTATION SCHEME FOR THE PARBEGIN-PAREND CONSTRUCTION

Florian Mircea BOIAN and Alexandru VANCEA*

Dedicated to Professor Emil Muntean on his 60th anniversary

Received February 25, 1994
AMS subject classification 68Q45, 68Q10

REZUMAT. - O schemă de implementare pentru construcţia PARBEGIN-PAREND. Lucrarea prezintă o schemă de translatare orientată spre sintaxă pentru construcţia PARBEGIN-PAREND, schemă pe baza căreia se poate construi uşor un translator care generează cod în limbajul C sub sistemul de operare UNIX.

The construction PARBEGIN $P_1 \mid P_2 \mid . \mid P_n$ PAREND [3] describes the simultaneous execution of the processes P_1, P_2, ,P_n and their parallel evolution until all of them terminate The n processes begin their execution at the same time and they function synchronously

This control structure contains a single entry (PARBEGIN) and a single exit (PAREND) and it is a *static* control structure, this meaning that all processing decisions are taken at compile time

The *fork-join* instructions are frequently used in UNIX, these being implemented by means of a *fork-wait* mechanism These instructions provide a direct mechanism for *dynamic process creation* and the possibility of multiple activations of the same process

The execution of a child process is made by calling the fork function which creates

* *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, 3400 Cluj-Napoca, Romania*

the child process by duplicating the father's image Fork returns in the father process the child's PID and zero in the child.

The UNIX fork-wait mechanism [2] allows the synchronization of a father process with its sons The wait function blocks the calling process until one of its childs terminates If at the moment of the call one of its childs it's already terminated the returning is immediate The value returned by wait is an integer representing the terminated child's PID

p = wait (&status)

where status is an integer providing information about the process status.

The synchronization with a certain child (let's say with the one having PID=pid1) can be done in the following way

while (wait(&status) != pid1),

These functionalities suggest the possibility of expressing a PARBEGIN-PAREND construction by means of the fork-wait mechanism

Let's consider the independent processes P_1, ,P_n as the subjects of a PARBEGIN-PAREND instruction, with the syntax

PARBEGIN P_1 PAR PAR P_n PAREND

(we introduced the word PAR instead of |, because the latter may be confused with the C bitwise OR operation)

In these conditions the PARBEGIN entry point has its equivalent in the sequence

```
                    if (fork()) == 0)  { P1; exit(0), },
    else    if (fork()) == 0)  { P2, exit(0), };
    else
else    if (fork()) == 0)  { Pn, exit(0), },
    else    for (i=1, i<=n, i++) wait(&status),
```

Having these, we can express the PARBEGIN-PAREND construction through the following syntax-directed translation scheme [1]

(1) <PARBEGIN_constr> .= **PARBEGIN process** <tail>,
if (fork()==0) {process; exit(0);} <tail>

(2) <tail> = **PAR process** <tail>,
else if (fork()==0) {process; exit(0);} <tail>

(3) <tail> · = **PAREND**,
for (i=1; i<=n; i++) wait(&status);

where we put the nonterminals between brackets

The **process** terminal designates one of the P_1, P_2, ,P_n processes

One of the issues that arise relatively to this scheme is how to handle nested PARBEGIN-PAREND constructs The answer is simple once the deeper construct has been identified and translated, it becomes a **process**

Production (1) will generate process P_1 The rest of the processes are generated by production (2), which also increments the number of processes by one Production (3) uses the number of processes for generating the PAREND waiting point correctly It's easy to write a translator for this mechanism

Let's see a generation example with two processes

(<PARBEGIN_constr>,
<PARBEGIN_constr>) ==>

(**PARBEGIN process** <tail>,
if (fork()==0) {process; exit(0);} <tail>) ==>

(**PARBEGIN process PAR process** <tail>,
if (fork()==0) {process; exit(0);} else if (fork()==0)
{process; exit(0);} <tail>) ==>

```
( PARBEGIN process PAR process PAREND ,
  if (fork()==0) {process; exit(0);} else if (fork()==0)
  {process; exit(0);} else for (i=1; i<=n; i++) wait(&status);)
```

REFERENCES

1 Aho A V , Ullman J D - The Theory of Parsing, Translation and Compiling, Prentice Hall, 1973

2 Rochkind M J - Advanced Unix Programming, Prentice Hall, 1985

3 Tanenbaum A S - Modern Operating Systems, Prentice Hall, 1992