Computer Science Journal of Moldova, vol.**2**, no.3(6), 1994

Concurrency and Portability

M.Evstunin

Abstract

Problem of software portability in concurrent environments are discused. A new method of logical partioning into indepedent level are supposed.

1 Introduction

Problem of software portability arised long ago but it did not lost it's topicality. A lot of difference platforms, computers, operating systems and environments increases the interest to this area. Software production turned into industrial phase. So, new technologies adequate for mass usage are necessary.

We can see the importance of this problem on example of Intel corp., which keep compatibility of 80x86 family expending a lot of resources. The millions of existing programs not permit to do step aside.

Many firms are specialized on producing and supporting multiplatform applications. We can divide them into two categories: producers of applications and producers of tools. We will deal with the problems of the second category.

During a number of years in S.-Petersburg University researches in area of creation and implemention of portable programming system based on Algol 68 have been carrying out. The problem posed for the author was formulated as follows: create a portable transput system for Algol 68. Algol 68 has concurrent computing facilities. This increases the complexity of porting process. Transput system itself is one of the most operating-dependent part of compiler. For this reason we had to develop sophisticated methods of logical partitionig into independent levels.

2 Concurrency Types

Let us at first fix the type of concurrency which we will deal with. There are two types of parallelism - multitasking and multiprocessing. In the first case process is considered as separate task from the operating system point of view. Tasks share the resources provided by OS, but they cannot access internal resources of each others. In the second case processes are started in such way that they share common resources of the parent which started them - common variables, memory, etc. (Fig. 1).

Take into account the following difference. In the first case times of life of processes are indepedent each from other, in the second case the death of parent implies the death of sons. We we'll deal with the second type of concurency only and with shared resources access. These restrictions are natural for our problem. For example, to implement a server part of client-server technology with few simultaneusly cooperating processes we can write

```
file system configuration;
proc(int num)int request handler;
par begin
        request handler(1),
        request handler(2),
        request handler(3),
        request handler(4),
        request handler(5)
    end
```

There are five concurrent cooperating processes in this example. They share access to common variable system configuration which has type **file** to control work. But new process can be started dinamically when necessary.

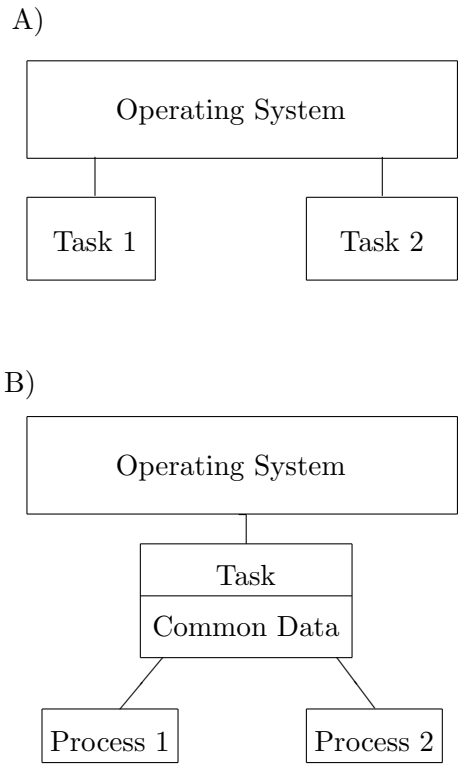

Figure 1: A) - multitasking, B) - multiprocessing

3 Logical Levels

The current transput system is based on J.C. van Vliet model [2]. The sophisticated treatment was done to elaborte criteria for logical division. Algol 68 has powerfull and flexible transput system described in [1, 2].

We'll not describe all procedures which are needed to implement this model (there are more 400 ones). Instead of we'll describe the logical levels.

1. Level of user interface. It consists of procedures which can be call by user from his program. This level is indepedent of operational environment.

2. Level of internal indepedence procedures. Here we can see procedures which aree called from user interface level or recursively by themselves. They don not call transput primitives (for example, convertion routines). This level is indepedent on operational environment.

3. Level of status file control procedures. It consists of routines which change transput mode, status of file, etc (for example, the procedures of changing mode from character to binary and vice versa, checking of possibility of binary transput, interpreting of format). This level is indepedent on operational environment but must be guarded from simaltaneous access.

4. Level of basic primitives. This level is indepedent of operational environment but must be guarded from simaltaneous access.

5. Level of memory managment. It includes procedures for allocating and deallocating memory for buffers, supporting lists of system resources, etc. This level is indepedent on operational environment but must be guarded from simaltaneous access.

6. Level of operating system interface. This is OS dependent level. This level must be rewritten during porting from one environmental platform to other. It takes from 8 up to 10 percents of

total number of procedures. It includes open and close file in the OS procedures, removing files, read-write buffers procedures, parsing file name one, etc.

Dividing software into such levels decrease effectiveness of program and increase portability. This is standard technique. We can use differing criteria and obtain differing results. We choose the criteria which help to increase independence on OS and to prevent simaltaneous access to shared resources. If we want to increse effectiveness on particular computer with particular OS we can to do inline substitution. Other methods of effectiveness increasing may be found in [3].

Cooperation of levels is shown on Fig. 2.

4 Shared resources protection

Let us to do a little remark. Usually user must prevent simaltaneos access to shared data himself. But there are a kind of errors type to detect and debug which is very dificulty. It is easier do not do them. So we made decision to provide guard of shared resources inside the transput system.

At least there are three variants:

- User provides guard of shared resources. This variant was discused above.
- Using greedy algorithm. In this case maximum part are protected. The procedures from the user interface turn into the following form:

```
proc(...)int  open =
begin
   down  guard;
   ...
   up  guard
end
```

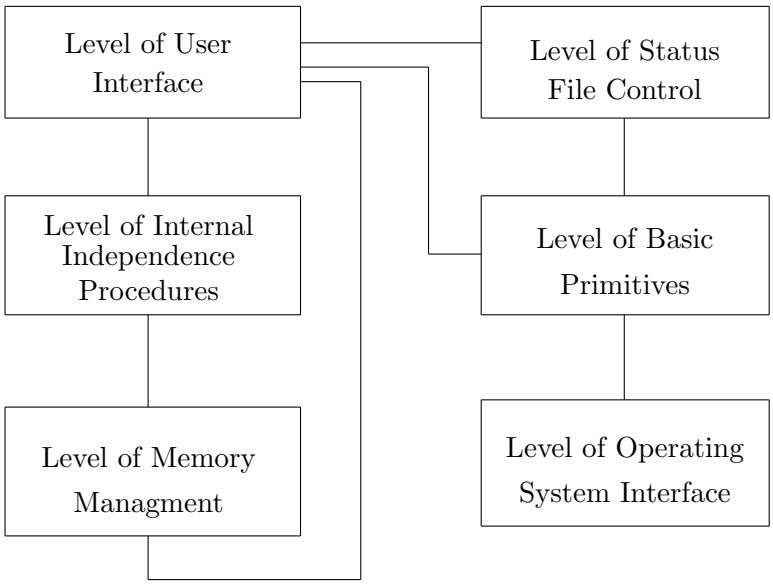

Figure 2: Levels cooperation.

The procedures of other levels are unchanged. This variant is the most simple for implementing. It is very close on sense to the guard which is provided by user. But the length of the protected parts is too long, so this deminishes the effectiviness of processes.

- An attemp to deminish overheads leads us to the third variant. Only those parts should be protected which are critical. Obviously, the procedures of levels of file status control, basic primitives and operating system interface have to be protected. If file

takes a necesary status nobody can change it until transput is finished. But processing of one file has not to block the processing of other ones. It entails the necessity of including the semaphore into file structure provided that it is unique for all copies of this object. The level of memory management should be also protected because procedures of this level provide support of lists of files, books, etc. This level protection is independent. Semaphore protection of all these programs has to be set before their calling because the recursive running of procedures has not to result in klinch state. This statement is applicable to any recirsive procedure.

Level of user interface and internal indepedence procedures don not need in guard.

Note that resources which are common for all processes must be guarded in any case (for example, lists of allocated and free memory, files, books, etc).

References

[1] Revised Report on the Algorithmic Lanuage Algol 68, Springer-Verlag (1975).

[2] J.C. van Vliet, Algol 68 Transput, part 1 and 2, Mathematical Center Track (1979).

[3] Sofware Portability, edited by P.J.Brown, Cambridge University Press (1977).

M.Evstunin,
Institute of Mathematics,
Academy of Sciences, Moldova
5, Academiei str., Kishinev,
277028, Moldova
e-mail: *evs@math.moldova.su*

Received 14 April, 1994