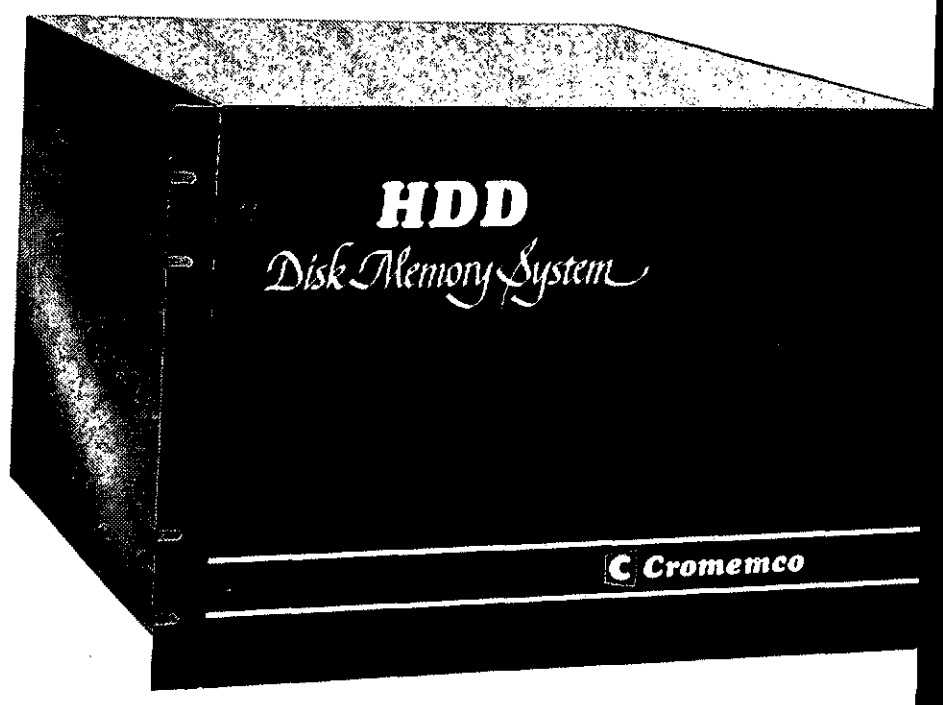
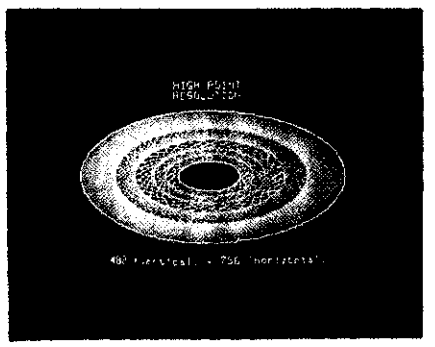
YU ISSN 0350-5596

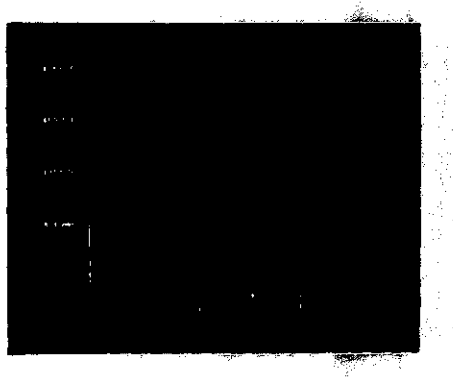
Cromemco
System Three Computer

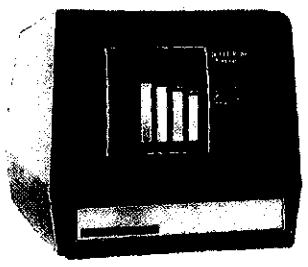
Mikro-kompjutor za profesionalnu upotrebu

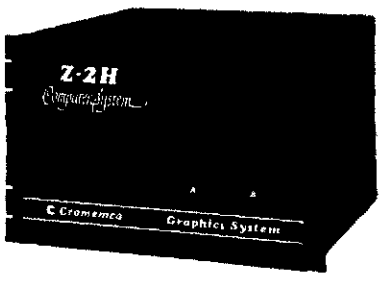


11 ili 22 Megabyte Winchester Hard disk



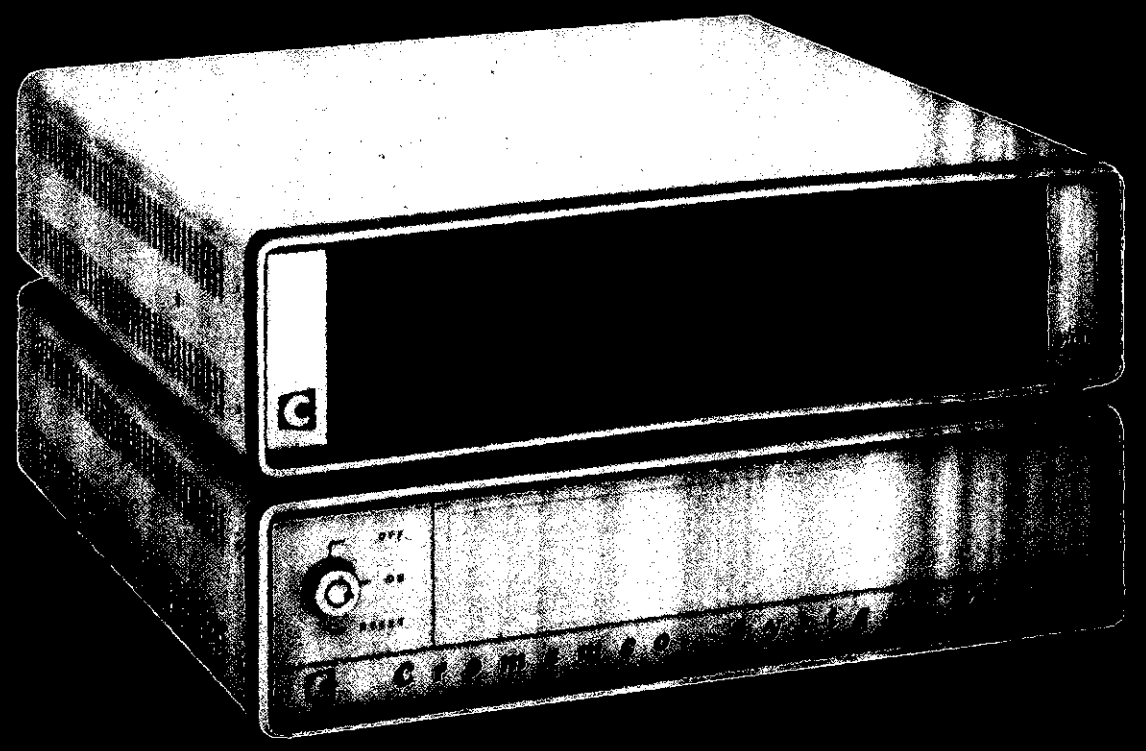






Kolor grafički sistem visoke rezolucije

System zero – namijenjen za specijalne aplikacije



Ekskluzivni zastupnik za SFR Jugoslaviju:

agromarketing

41000 Zagreb, B. Adžije 7/1, P.P. 5
Telefon: (041) 417 662, telex: 21741

informatica

Časopis izdaja Slovensko društvo INFORMATIKA, 61000 Ljubljana, Parmova 41, Jugoslavija

Uredništvo in uprava: Informatica, Parmova 41, 61000 Ljubljana, telefon (061) 312-988, teleks: 31366 YU DELTA

Letna naročnina za delovne organizacije je 500,00 din, za redne člane 200,00 din, za študente 100,00/50,00 din, posamezne številke 100,00 din
Žiro račun št.: 50101-678-51841
Stališče uredništva se lahko razlikuje od mnenja avtorjev.

Pri financiranju revije sodeluje tudi Raziskovalna skupnost Slovenije.

Na podlagi mnenja Republiškega sekretariata za prosveto in kulturo št. 4210-44/79 z dne 1.2.1979, je časopis oproščen temeljnega davka od prometa proizvodov.

Tisk: Tiskarna KRESIJA, Ljubljana

Grafična oprema: Rasto Kirn

ČASOPIS ZA TEHNOLOGIJO RAČUNALNIŠTVA IN PROBLEME INFORMATIKE
ČASOPIS ZA RAČUNARSKU TEHNOLOGIJU I PROBLEME INFORMATIKE
SPISANIE ZA TEHNOLOGIJA NA SMETANJETO I PROBLEMI OD OBLASTA NA INFORMATIKATA

YU ISSN 0350–5596

LETNIK 5, 1981 – št. 2

V S E B I N A

informatica

Published by INFORMATIKA, Slovene Society for Informatics, 61000 Ljubljana, Parmova 41, Yugoslavia

Headquarters: Informatica, Parmova 41, 61000 Ljubljana, Phone: (061) 312-988, Telex: 31366 Delta

Annual subscription rate for abroad is US $ 22 for companies, and US $ 7,5 for individuals.

Printed by: Tiskarna KRESIJA, Ljubljana

DESIGN: Rasto Kirn

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350-5596

VOLUME 5, 1981 — No. 2

CONTENTS

INFORMATICA 2/1981

ISSUES IN LARGE PROGRAM DESIGN AND IMPLEMENTATION

WŁADYSŁAW M. TURSKI

UDK: 681.3.06.002.2

INSTITUTE OF INFORMATICS WARSAW UNIVERSITY POLAND

After analysing the notion of large program and its various ramifications, the time-evolution of specifications is singled out as their most important characteristics. Some technical consequences of the acceptance of the possibility to change the specifications are described and a framework in which relationships between an application, a program and a specification can be uniformly treated is presented.

The notion of "large program" is by far more often used than explained and/or defined. In fact, one suspects that this notion is but a convenient catch-all for a quite diverse class of software, the shared attribute of its members being a peculiar mixture of technical, managerial and political problems involved in their design, programming, implementation and continuing application (cf./16/,/17/). Frequently quoted examples of large programs include: operating systems for mainframe computers, software systems for banks and insurance companies, software support systems for space projects, complex inventory/supply systems (such as used by large companies and armies) and a variety of software systems used by the defence establishments.

That the ill-defined class of large programs is a subject of great and fast growing interest to a significant number of information-processing professionals should come as no surprise: regardless of individual differences, the shared characteristics of large programs translate into a univocal financial equivalent: large programs are very expensive to make, and even more so to maintain. Thus, each technique, procedure or method which promises to save let even only a few percent of large program costs is a wellcome addition to the ever growing body of the professional folklore. Few percent of a billion (US) dollars is a sum not to be treated lightly even these days of a double-digit inflation. (I do not know how large a fraction of the total investment in software is attributed to large programs, nor it is possible to determine this figure precisely in view of a rather diffused distinction between a large and a non-large program. Since, however, the costs of known specimens of obviously large programs range from 10^6 to 0.5×10^9 US$, it is safe to assume that the world-wide investment in large programs is of the order of 10^{10} US$ and increases by some 10^9 US$ each year, cf. the data quoted in /13/)

Thus, confronting the large program issue we are looking at big money, at an industry mass-producing expensive products, and - to the best of my knowledge - at the only big industry based primarily on the resources of human intellect (the raw material and energy consumption required to make, install, operate and service a large program is practically nill if the energy is measured in horsepower; the situation could be different if the energy was measured in units of human brain power, but the latter are neither defined nor socially acceptable).

It is not surprising, therefore, that the motivations behind the research on large programs are not only derived from but also couched in terms of managerial concerns: costs, productivity, manpower, deadlines, scheduling etc. While quite legitimate from the industrial point of view, these concerns tend to obfuscate the technical problems involved in the design and implementation of programs falling into the considered class. In some instances, an inappropriately chosen managerial term not only distracts from the underlying technical problem but also creates an utterly artificial one, starting a frantic chase after a phantom, after a solution of a non-problem. (One of such pseudoproblems, that of the software maintenance, has been analysed in /17/ and shown to consist in a misunderstanding: instead of "maintenance", the actual goal should be "stable evolution" of software.)

In the present paper we shall consider some technical issues of large program design and implementation, and, where appropriate, indicate their managerial consequences.

Having investigated a number of large programs throughout their life-cycles, Belady and Lehman /2/ have formulated a number of "laws" capturing the phenomenological properties of the time-evolution of large programs. The essence of the Belady-Lehman laws consisted in stating that each large program either undergoes a continuous change or becomes gradually less and less useful, that the rate of change - in the long run - cannot be arbitrarily controlled by managerial parameters (such as putting more people on the project) and that certain global characteristics of an evolving large program, such as its growth rate, are, statistically speaking, invariant throughout its life-span. Arguably the most important of the "laws" states that the complexity of a large program increases with its age unless an effort is made to control it.

The significance of Belady-Lehman investigations was twofold:

(i) they have established that many different instances of the class "large programs" have similar measurable properties and that, therefore, these properties could be accepted as the defining properties of the heretofore fuzzily defined class; and

(ii) they pointed out that naive managerial controls are inapplicable to the large program evolution, indeed that quite often they are simply counterproductive.

(As an illustration of the second point consider a typical managerial reaction to the, unfortunately, quite typical situation arising in large software projects: a decline in the "productive output" of the team engaged in programming, perhaps leading to an overrun deadline. The naive reaction is to put more programmers on the project. The actual result of this decision is ... a further decrease of the "productive work rate": daily "output" instead of picking up drops down, sometimes quite dramatically. Since the newcomers have to be integrated into the original team, informed about the project and about their assignements, and the task of bringing them up to date is given to those already on the project, for a non-negligible period of time the "old hands" have to carry a double burden: that of educating new colleagues and that of keeping up with work, whereas the new additions to the project task force are totally "non-productive". Of course, after some time the newcomers are properly meshed into the team and start "producing". Unfortunately, by that time the delay has grown and the management might be tempted to increase the number of programmers even more... Even if this does not happen, the increase in the number of people has combinatorially increased the internal communication problems unless a great care had been exercised to avoid the growth of complexity. Because spending good money on academic pursuits, such as the reduction of complexity is seldom viewed by the management as an attractive proposition, the grown complexity reduces the potential productivity of the team far below that what could be expected and naively extrapolated from the measurements made when the team had its original size.)

The use of time-development characteristics for the definition of the class of large programs, although possible, was found to be somewhat unsatisfactory from the methodological point of view. If applied, it would amount to saying: large programs are those that evolve in such and such way; the discovered regularities would not be "laws" any more, reduced to mere defining properties they would be all too easily swept aside by saying: "well, my program is large, but not of this evolving variety". In fact, having accepted the Belady-Lehman laws as definitions, we would find ourselves inside a vicious circle rather than in a universe of large programs.

In /13/ Lehman significantly modified his approach, instead of speaking about large programs in general, he introduced an interesting classification, in which several classes of programs were distinguished according to their relationship to the specifications and to the models of the real world used in formulating the specifications. The time-evolution is now considered as a property enjoyed by programs whose specifications change in time, either because their author has changed his perception of the real world, or because the application of the program has changed the real world to such an extent that the change in program specifications is required.

(Consider, for example, a software system supporting operations of a bank. Its original specifications were based on the banking regulations and on the observation of customers' behaviour in a non-computerised banking environment. When computerised, or partially computerised teller-stands are introduced some patterns of clients' behaviour change and thus correspondingly change the requirements and specifications for the supporting software. The computerisation of the internal banking procedures and, particularly, the introduction of "instant banking" by means of telecommunication and information processing technologies, create entirely new possibilities that change not only the customers' behaviour patterns but also lead to certain problems necessitating changes in the banking regulations. These, in turn, cause revisions of system software specifications.)

Hence we are faced here with a major technical problem: how to proceed about making programs according to changing specifications. I believe this is the only major technical problem that could be specifically attributed to the class of large programs. The problems arising from the sheer volume of program texts, if considered in an environment in which the specifications are fixed and guaranteed not to change, are - at least in principle - quite well-understood and do not require any special treatment beyond disciplined application of modular programming techniques /14/ on the substance-matter side, and a consistent managerial set-up on the organizational side (such as provided by, e.g., the chief programmer team structure /1/). Even if the specifications convey incomplete information about the desired program, if we may assume that no additions will ever be made to the original specifications, the problem of making a large program does not fundamentally differ from that of making a non-large program; in fact, the "size" of the program is merely a consequence of whatever solid information can be gleaned from the specifications, and, naturally, of our ability to program well.

True enough, when the real-world problem, as captured by the specifications, requires an extended and complicated description in which a great deal of irreducible notions are employed and a complex relational structure is imposed on these notions, the programming process may justifiably be split into several stages, more formally separated one from another than is the case when standard techniques of separation of concerns /8/ and stepwise refinement /19/ can be, as if it were, informally interwoven with program text development. (Typical of the methods that formalize the separation between the stages of programming is the "programming-in-the-large" technique /9/, /15/, which formalizes the process of deriving the overall structure of the program on the modular level, to be followed by detailed programming on the intramodular level.) It is also true that in some application-oriented environments it was found advantageous to use a set of rules providing step-by-step directions how to transform the original specifications (imprecisely stated or expressed in terms poorly related to the programming activities) into a more manageable form (such as SADT, Jackson Method, HIPO charts and procedures). Without, however, in the least belittling the importance of such techniques, indeed, while strongly recommending their use (if for no other

than certainly for their salutary disciplinig effect, both on the management and on the programmers, and for the support they provide in clerical aspects of program development), we must be aware of the fact that such methods rely heavily on the immutability of the original goals (if not the specifications) and thus do not necessarily help at all when as a result goals modification the specification changes too.

Hence, as we have already remarked, the real problem of the considered class of programs is how to cope with changing specifications.

First, let us consider why changing specifications present such a difficult problem for software development?

All known programming techniques and methods accept the fundamental criterion of software quality: programs must be first of all correct. While the the exact meaning attributed to the notion of program correctness may vary a little /3/ depending on, e.g., whether we are concerned with partial or total correctness, or even in some cases on the subject matter (for instance, the notion of correctness of data base programs may include explicit references to the integrity of data, usually neglected or implicit in similar notions for other types of software), the crux of the matter stays always the same: correctness is defined and understood as a relation between program and its specifications. The exact form of this relation reflects to certain degree the preferred point of view, but there is no other way to introduce a meaningful notion of correctness.

Thus, if we start programming taking a specification S0 as the statement of objectives to be achieved and produce a program P0 such that the pair (S0,P0) satisfies the particular relation C that represents our chosen concept of correctness: C(S0,P0) = "program P0 is correct with respect to specification S0", then any change of S0, say to S1, may invalidate the established relationship: C(S1,P0) needs not to hold. The fallacy of the "software maintenance" rests in the belief (usually implicit) that program P1 such that C(S1,P1) can be obtained from P0 by relatively simple operations performed on the text of program P0. The insidiousness of this fallacy consists in that "relatively simple" is a very elastic term. Since a total reprogramming, i.e. discarding the text of P0 and writing a totally new program P1 may be accepted as an extreme case of "relatively simple operations on the text of P0", the theoretical validity of the software maintenance approach can not be questioned: as long as the specification S1 is not inconsistent, a suitable P1 can always be written! But it is not necessarily true that P0 is a good approximation to P1.

The key to the methodological solution of the changing specifications problem is to be found in thorough investigations of the specification/program relationships. In a series of papers /16/,/17/,/18/ it has been proposed to consider programs as models of specifications, where the term "model" is understood as in mathematical logic; consequently, specifications are understood as theories. The view that specifications are theories is also accepted in /6/,/7/, where algebraic (categoric) operations on theories are used as a means of expressing extensions and other modifications of specifications stated in a special language Clear.

If specifications are to be considered as theories of which programs are models, it becomes necessary to formalize the specifications in the sense that they must be expressed in a well-defined formal system (this is not equivalent to saying that specifications must be written using mathematical symbols!).

The need to formalize the specifications seems to be gaining a universal appeal. In the second half of 1970s several major projects were undertaken aiming at the construction of suitable tools, primarily - special languages; for a survey of results see /10/, /4/,/5/,/12/.

The general interest in formal specifications has arisen from two sources:

(i) algebraic specification of data types, a very fruitful area of research opened up by the now famous Guttag's Ph.D. dissertation /11/,

(ii) formal definitions of programming languages, first undertaken by tha Vienna Laboratory of IBM /20/.

Most of the work on formal specifications, however, finds - as yet - little appreciation in the day-to-day practice of programmers, or, more generally, in software development. One of the reasons for the slow acceptance of formal specification methods can be traced to a pretty well-spread reluctance of the programmers, not to mention the so-called system analysts, to apply any mathematically flavoured techniques. Another reason is a deep-seated misconception that the language of mathematics is too dry, too abstract to express the richness of the real-world problems. As a consequence, a variety of semi-formal techniques have been presented and at least some of them were reasonably well received by the practitioners in the field (as wittness the commercial success of SADT method marketed by SofTech).

The practical success of semi-formal specification techniques is quite significant from the methodological point of view as it indicates a fast growing yearning for specification tools more precise than the plain language. In fact, when programmers shrink away from the natural language specifications, it is not so much the actual form of the specification that is found unacceptable, as the unlimited scope of misunderstanding left by the use of a language with non-formalized semantics. To understand a statement expressed in a natural language requires that a correct context be established, which seldom can be achieved by purely algorithmic means. It usually depends on an implicit assumption of deep understanding of the environment, an understanding shared by the author of the specifications and by the programmer.

The real advantage of formalized specifications over informal ones rests, of course, not in the use of symbolics, but in not assuming anything, in not taking for granted that the author of specifications and the programmer had any common understanding of words. The paucity of formal means of expression is beneficial in itself: it makes people to state explicitly even the most obvious, which frequently is not so obvious after all.

That the "pictorial" formalism of some of the semi-formal specification techniques (SADT is again a good example) should be more readily acceptable than the more tractable mathematical ones is a sad reflection on the educational systems; it is, however, only a matter of time before the greater flexibility and,

first of all, the conciseness of formal systems will speak for itself, just as it happened in programming, where flow charts have been replaced by structured presentations of linguistic forms. I am pretty certain that the use of formal specification techniques will spread and, eventually, it will seem just as natural as the use of programming (formal) languages is today considered natural for the description of algorithms.

Formalization of specifications is a prerequisite for a systematic treatment of specification/program relationship. Indeed, the well-known techniques of structured programming, i.e. the separation of concerns and stepwise refinement, closely related to practical aspects of designing and implementing correct programs, can be easily represented in the framework created by the formalization of this relationship /16/, /18/. But formalization of specifications alone is not enough if we want to control the phenomena of program evolution - the most important practical aspect of large program design and implementation.

In order to cope with program evolution driven by specification changes we must investigate such formalisms which allow to express not only the specifications but also their changes. This, in turn, immediately raises two further problems:

(i) how to determine the class of possible modifications to be considered with a given specification?

(ii) how to achieve the stability of program evolution, i.e. such a situation, in which "small" modifications of the specification would require "small" changes of the program?

Even though these two problems could be tackled separately, the most promising line of attack seems to consist in approaching them jointly, in a somewhat pragmatic fashion.

Let us first observe that neither a concrete specification, nor its possible modification arise in vacuum; even if a very abstract formal system is used as a means of expression, the content of the specification is related to a specific problem or need experienced in the physical world. It is often said that the specification for a piece of software is abstracted from a less formal description of a problem or even from a general knowledge of an application domain. While this is certainly true in the sense that the need to apply a computer arises in the course of some specific activity, the vagueness of the notion of "abstracting from the general knowledge" makes it quite useless for any practical purposes.

Instead, we propose to consider the relationship between the specification and the application domain exactly like the relationship between the specification and the program domain, i.e. to treat the specification as the theory of the particular application.

From the formal point of view we are exploiting the well-known fact that theories can have different, non-isomorphic models; in our case: in program domain and in application domain. (The non-isomorphism of models is a slight departure from standard mathematical preferences, but this should not disturb us.) From the practical point of view we are making a twofold request:

(i) the application domain should be represented in an application language (just as the program is represented in a program language),

(ii) it should be proved that teh particular application satisfies the specification (just as we are obliged to prove that a particular program satisfies the specification).

The design of the application-domain language is exactly this activity in which the general knowledge of the application is useful, indeed, necessary. It should be noted that the application-domain language needs not to bear any resemblence to programming languages, its construction may be left entirely to the application experts. Moreover, there is also no need to provide any translation between the application and programming languages since the relationship between a concrete application and a concrete program is not that of translation, but an entirely different relationship, viz. that of being models of the same theory (specification).

The specification, written in a formal system different from both programming and application languages, is a theory for which at least two models should be constructible: one - in the application domain, another - in the application domain. The program model of the specification is constructed by "normal" programming techniques. In principle, one could insist that the application model should be similarly constructed. It seems however much more useful to have the application model constructed independently of the specification and then verify the specification by checking if the constructed model indeed satisfies the specification. If not, the rational thing to do is to change the specification. Of course, if this kind of change in specification is foreseen it does not make sense to even start programming before the specification is verified!

When the specification has been verified against the application model (formally speaking, the application is found to be a model of the specification, but in pragmatic terms we have established that the specification is a good theory of the application), we may safely construct a program; if correctly made, it will "behave" exactly as expected. Should now the expectations change we may insist that the new requirements be phrased not as direct requests to modify the program (as is by far too frequently done today), but as suitable modifications of the application model. If the modified application model still satisfies the original specification-theory there is no need to change the program, if - as is more likely to happen - the specification is not satisfied any more by the modified application model, before any programming activity is undertaken, the satisfaction relationship between the specification and the application model has to be re-established by suitable modification of the former.

The ease (or difficulty) of re-establishing the required relationship between the application model and the specification may serve as an indicator of the magnitude of programming task to be accomplished. Insisting that the re-establishing be done with full rigour may prevent ill-conceived requests to modify the existing software (it would eliminate, for instance, all sorts of inconsistent requests).

Another advantage of considering the application/specification relationship rigorously and thoroughly before any programming is undertaken refers to the possible future modifications. It is quite likely that in building the application model some arbitrary decisions will be made. Constructing the theory

of the application (i.e. the specification) we can consider the differences between the adopted and rejected variants and study corresponding differences in specification. Thus, at least in some instances, the specification may be annotated with alternatives informing about changes that will be required should the application experts change their mind. Since quite often the modification of the application model consists in just accepting such second thoughts about previously rejected alternatives, a wise programmer will cater for such contingencies by constructing a program in which it would be easy to switch to another variant (as a minimum, programming decisions making such switch unnecessarily difficult could be avoided).

Finally, for a given specification formalism , some typical specification-changing operations may be discovered such that corresponding program model modifications would be algorithmically defined. A catalogue of these operations should accompany a concrete specification and a library of corresponding procedures should be provided with a software product. When a change is required it could be analysed with respect to the catalogue: any change which can be effected by a legal combination of catalogued operations can be modelled in the program domain by performing a suitable sequence of library procedures. Specification changes amenable to such treatment certainly belong to the class of small changes with respect to which the software product is stable.

It is a well-known fact that practical models of theories are usually richer than their theories in the sense that they can have properties not captured by the theory. Quite naturally, programs also have properties not prescribed by their specifications (a sorting program may, e.g., be specified only with respect to its net effect, its time- and space-complexity being left unspecified). Thus it is perfectly possible to have different program models of the same specification and it is a programmer's prerogative to exploit the freedom left to him by the specification. It is because of this freedom that programmers can try to make "as efficient as possible" programs. Taking into account, however, that the specifications of large programs may change with time, an extensive exploitation of the freedom left by the initial specification may turn out not to be such a good programming practice after all: properties left unspecified by the initial specification may be fixed in one of the forthcoming changes differently from the way in which they were fixed in the program, thus causing perhaps a quite extensive change in the implemented program. There is no method that could prevent this to happen, but a programmer who expects the specification to change by freezing an initially unspecified aspect should try to avoid making use of his (legitimately) arbitrary decision at too early a stage of programming, and certainly should preserve the solution obtained on this level of abstraction at which such a decision was not yet made: when the specification change freezes a heretofore free property, he needs only to backtrack to this level to regain the ability to satisfy the changed specification.

REFERENCES

1 Baker, F.T.: Chief programmer team management of production programming. IBM Syst. J. 11 (1972), 56.

2 Belady, L.A. and Lehman, M.M.: A model of large program development. IBM Syst. J. 15 (1976), 225.

3 Berg, H.K., Franta, W.R., Moher F.G.: Correctness of software - an overview. In /4/.

4 Berg, H.K. and Giloi, W.K. (Eds.): The Use of Formal Specifications of Software. Informatik-Fachberichte 36 (1980), Springer-Verlag.

5 Bjørner, D. (Ed.): Abstract Software Specifications. Lecture Notes in Computer Science 86 (1980), Springer-Verlag.

6 Burstall, R.M. and Goguen, J.A.: Putting theories together to make specifications. In Proc. Fifth Int. Joint Conf. on Artificial Intelligence, Cambridge, Mass. (1977).

7 Burstall, R.M. and Goguen, J.A.: The semantics of Clear, a specification language. In /7/.

8 Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall, 1976.

9 DeRemer, F. and Kron, H.: Programming-in-the-large versus programming-in-the-small. In Proc.Int. Conf. on Reliable Software, Los Angeles (1975).

10 Freeman, P.: Software design techniques: current options. In Software Development Techniques, The Seventy-Second Infotech State of Art Conf., London (1980).

11 Guttag, J.V.: The specification and application to programming of abstract data types. Univ. of Toronto, Dept. of Comp. Sci. Rep. CSRG-59 (1975).

12 Jones, C.B.: Software Development, A Rigorous Approach. Prentice-Hall, 1980.

13 Lehman, M.M.: Programs, life cycles, and laws of software evolution. Proc.IEEE 68 (1980), 1060.

14 Parnas, D.L.: A technique for software module specifications with examples. Comm.ACM 15 (1972), 330.

15 Turski, W.M.: Computer Programming Methodology. Heyden, 1978.

16 Turski, W.M.: Design of large programs. In Software Engineering - Entwurf und Spezifikation, C.Floyd und H. Kopetz (Hrsg.), Teubner, 1981.

17 Turski, W.M.: Software stability. In Systems Architecture. Proc. 6th ACM European Regional Conf., London (1981).

18 Turski, W.M.: Specification as a theory with models in the computer world and in the real world. In System Design, P.Henderson (Ed.), Pergamon-Infotech, (to appear).

19 Wirth, N.: Program development by step-wise refinement. Comm.ACM 14 (1971), 221.

20 Wegner, P.: The Vienna definition language. Comp. Surveys 4 (1972), 5.

INFORMATICA 2/1981

PROGRAMIRANJE SPROTNIH IN VGNEZDENIH SISTEMOV: PROCESI V ADI

MATIJA EXEL ,
FRANČEK PRIJATELJ

UDK: 681.3.06 Ada

INSTITUT JOŽEF STEFAN

Članek predlaga opredelitev sprotnih in vgnezdenih sistemov. Nato prikaže razvoj in glavne značilnosti Ade, jezika za programiranje takih sistemov. Osrednji del članka obravnava definicijo procesov v Adi, opisuje mehanizme za medprocesno sinhronizacijo in komunikacijo v Adi ter navaja primere uporabe procesov.

The programming of real-time and embedded systems: processes in Ada.

The introductory part of the paper proposes a characterization of real time systems and of embedded systems. The language Ada is the most recent proposal for the programming of such systems. The development and main features of Ada are considered. The main part of the paper describes the Ada tasks: definition, mechanisms for task synchronization and communication, examples.

1. UVOD

V uvodu bomo opredelili položaj jezika Ade glede na druge obstoječe visokonivojske programske jezike in glede na spekter aplikacij, ki naj bi jih pokrival.

Jezik Ada je prvinsko namenjen za programiranje sprotnih vgnezdenih računalniških sistemov. Prvi del uvoda bo torej na kratko podal opredelitev sprotnih in vgnezdenih računalniških sistemov, drugi del uvoda pa bo podal razvojno zgodovino Ade in njene glavne značilnosti.

V nadaljevanju se bomo omejili na obravnavanje procesov v Adi: pokazali bomo, kako Ada rešuje medprocesno komunikacijo in sihronizacijo in kakšne pripomočke nudi za lažje programiranje sprotnih sistemov.

1.1. Sprotni sistemi in vgnezdeni sistemi

Opredelimo računalniški sistem kot kombinacijo hardvarskega sistema (računalnik(i), standardna računalniška periferija) in softwarskega sistema (operacijski sistem, aplikacijski sistemi,...).

Glede na okolje računalniških sistemov lahko slednje opredelimo kot neodvisne ali vgnezdene. Okolje neodvisnega računalniškega sistema so samo ljudje-uporabniki; okolje vgnezdenega računalniškega sistema pa vsebuje še druge naprave, ki niso računalniške narave. Lahko bi rekli, da so vgnezdeni računalniški sistemi podsistemi splošnih sistemov. Splošni sistem z vgnezdenim računalniškim sistemom je lahko opredeljen kot sistem sodelujočih neračunalniških naprav, za katere igra vgnezdeni računalniški podsistem vlogo koordinatorja, nadzornika oz. regulatorja. Primer splošnega sistema je npr. raketa, pralni stroj, avtomatiziran industrijski kompleks, telefonska centrala, laboratorijska instrumentacija, avtomobil, sistem kontrole zračnega prometa, avtomatska telefonska centrala, numerično kontroliran stroj; v vseh teh sistemih je lahko (mikro)računalniški sistem udeležen kot podsistem oz. je v njih vgnezden.

Pojem sprotnega (real-time) sistema je precej širok in v literaturi tudi ni točneje opredeljen oz. definiran.

Glavni faktor sprotnosti je čas: računalniški sistem, neodvisen ali vgnezden, se mora odzvati zunanjim dogodkom v določenem roku, kar implicira, da mora biti trajanje določenih računalniških obdelav omejeno. Lahko torej rečemo, da je računalniški sistem sproten, kadar na kakršenkoli način upošteva dejstvo, da je odzivni čas na zunanje dogodke omejen.

Lastnost sprotnosti računalniškega sistema je relativne narave: računalniški sistemi so lahko "več ali manj" sprotni. Lastnost sprotnosti lahko opredelimo z naslednjimi kriteriji:
- z zahtevanimi odzivnimi časi (kratki časi implicirajo "večjo" sprotnost);
- z mero kritičnosti nespoštovanja odzivnih časov. Nespoštovanje odvisnih časov lahko povzroči večjo ali manjšo škodo; posledice nespoštovanja teh časov so lahko raznovrstne: od neugodja operaterja na terminalu s predolgim odzivnim časom, preko gospodarske škode, ki jo povzroča neprimerno krmiljen splošni sistem, pa do katastrof (npr. izgube življenj, uničenje naprav), ki jih lahko povzroči nezadostno sproten računalniški sistem, vgnezden npr. v industrijskem

kompleksu;
- s striktnostjo spoštovanja odzivnih časov: ta kriterij se navezuje na mero kritičnosti. Kadar je kritičnost velika, je potrebno računalniški sistem organizirati tako, da so odzivni časi vedno spoštovani oz. realizirani; kadar kritičnost ustreza večji ali manjši škodi, ki je lahko za krajši čas oz. v določenih okoliščinah tolerirana, potem si računalniški sistem v ustreznih situacijah lahko dovoljuje začasno nespoštovanje odzivnih časov.

Čeprav se je pojem sprotnosti verjetno sprva pojavil kot karakteristična zahteva nekaterih komercialnih aplikacijskih sistemov (npr. sistem rezervacije letalskih kart), ki so neodvisni računalniški sistemi, pa je jasno, da so in bodo zahteve sprotnosti bolj tipične in "večje" za vgnezdene računalniške sisteme.

1.2. Jezik Ada

1.2.1. Pomen in razvoj Ade

Jezik Ada, kot smo že omenili, je bil zasnovan za programiranje sprotnih vgnezdenih računalniških sistemov. Glede na obstoječe visokonivojske programske jezike predstavlja revolucijo iz naslednjega gledišča:
- je sistemsko-aplikacijski jezik, ki uvaja oz. združuje nove jezikovne mehanizme za tako zvano "programiranje v velikem" (programming in the large), ki zadeva sestavljanje in kombiniranje skupine programov v celotne sisteme.

Predno si podrobneje ogledamo razvoj Ade in zahteve, ki so bile Adi zastavljene, poglejmo, kaj zgoraj omenjena revolucija pomeni.

Jezikovno sceno so, do prihoda Ade, zasedali v glavnem jeziki, namenjeni za programiranje aplikacij, kot so Fortran, Algol, PL/1, Cobol, Pascal. Ti jeziki so prirejeni neodvisnim računalniškim sistemom, v katerih je programska oprema pretežno ločena na sistemsko programsko opremo (operacijski sistemi, jezikovni sistemi (prevajalniki in druga razvojna oprema), razni podporni sistemi,...) in aplikacijsko programsko opremo. V teh sistemih sta se uporabljali pretežno dve kategoriji programskih jezikov: sistemski jeziki (često zbirniki) za programiranje operacijskih sistemov in aplikacijski jeziki za programiranje aplikacij. Do te razdelitve je prišlo, ker se je predpostavljalo, da so aplikacije povsem drugačne narave od operacijskih sistemov in da je možno neodvisne računalniške sisteme prirediti različnim namenom s programiranjem aplikacijske programske opreme, nadgrajene razmeroma standardnim tipom operacijskih sistemov. Od tod torej razlikovanje med sistemskimi in aplikacijskimi jeziki.

Slika je povsem drugačna pri vgnezdenih sistemih, posebno, kadar so ti močno sprotne narave. Tu se pokaže, da ločitev na sistemsko in aplikacijsko opremo nima več veliko smisla iz dveh razlogov:
- vgnezdeni sistem je normalno tesno prilagojen konkretni aplikaciji, kar pomeni, da operacijski sistem ne igra več vloge skupne osnove za različne aplikacije, temveč da je tudi ta tesno prirejen aplikaciji. Programski sistem je torej zaključena celota, prirejena potrebam vgnezdenega sistema;
- vgnezdeni sistem je normalno sprotne narave. Sprotnostne zahteve so navadno tako specifične, da zahtevajo specifičen operacijski sistem, ki mora biti poleg tega zelo tesno navezan na aplikacijsko nadgradnjo. Od tod potreba po zlitju operacijskega sistema in aplikacijske nadgradnje v koherentno, zaključeno celoto (glej 7).

Posledica zgornjega je torej potreba po skupnem sistemsko-aplikacijskem jeziku, ki naj omogoča koherentno in poenoteno programiranje celotne programske opreme vgnezdenega sprotnega sistema. Jezik Ada naj bi odgovarjal tej potrebi.

Do prihoda Ade se je za programiranje sprotnih in vgnezdenih sistemov uporabljala cela vrsta jezikov: Algol 68, Pascal, CAMAC-IML, PL/1, Cobol 66, Procol, Pas1,RTL/2,Modula(9), Concurrent Pascal(13), Jovial (razne verzije), CMS-2, TACPOL, SPL/1, itd. Ti jeziki so ali neprimerni za današnje potrebe ali pa so zastareli. Nujno je bilo torej podvzeti akcijo za izbor oz. izdelavo standardnega jezika, ki bi pokrival celotno področje sprotnih vgnezdenih sistemov. Pri tem sta bila aktivna:
- LTPL-E komite, ki deluje v okviru institucije "International workshop on industrial computer systems" ter
- ministrstvo za obrambo ZDA.

Ministrstvo za obrambo ZDA je leta 1974 zastavilo zelo širokopotezno akcijo za specifikacijo zahtev in zmožnosti takega jezika. Na podlagi teh zahtev je bilo narejeno večje število predlogov za programski jezik (v ožjem izboru so bili štirje jeziki, ki so vsi temeljili na Pascalu, čeprav so bili priporočeni trije temeljni jeziki: Pascal, PL/1 in Algol 68). Končno je bil sprejet predlog za jezik imenovan Ada (po Adi, Byronovi hčerki in Babbageovi asistentki; slovi tudi kot prvi programer). Po preliminarnem priročniku (1979) je bil lani izdan standardni, do nadaljnega definitivni priročnik (ref. 1.).

1.2.2. Značilnosti Ade

Značilnosti Ade izvirajo iz zahtev, ki so bile zastavljene:
1)- zanesljivost in varnost programske opreme;
2)- zmožnost za programiranje celotnih (zaključenih) sistemov: modularizacija, vzdrževanje (programiranje v "velikem");
3)- zmožnost realizacije sprotnostnih zahtev;
4)- zmožnost realizacije vgnezdenih sistemov.

Iz zgornjih zahtev sledijo naslednje značilnosti:

1)- močno tipiziranje podatkov (zahteva 1): to omogoča, da že ob času prevajanja preverjamo pravilnost uporabe podatkov na podlagi tipskih informacij;

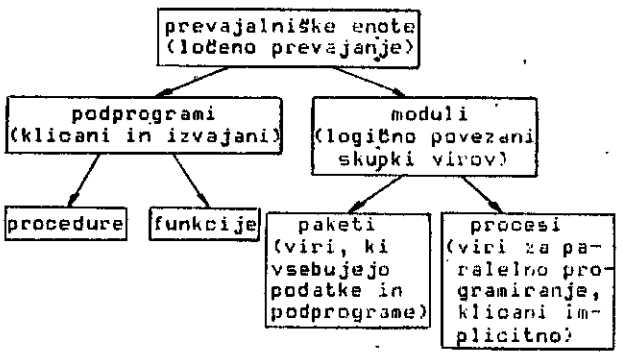
2)- modularnost in vidljivost (visibility) ter ločeno prevajanje (zahtevi 1,2); moduli so lahko enote:
- vidljivosti (glej Modula, ref.9) oziroma dosegljivosti podatkov (primerjaj s klasično bločno strukturo);
- abstrakcije (algoritmične in podatkovne);
- ločenega prevajanja;
- asinhronosti (posli, procesi).

Na tem področju je Ada privzela nekatere

koncepte jezikov, kot so Euclid (10), Lis, Mesa (8), Modula (9), Sue, Alphard (6) in CLU (11). V Adi so torej prisotne programske enote, ki ustrezajo vsem zgornjim kategorijam modulov:

- podprogrami, paketi in procesi so prevajalniške enote (ločeno prevajanje);
- podprogrami (funkcije, procedure) so algoritmične abstrakcije;
- paketi (skupki podatkov in podprogramov) so podatkovne abstrakcije;
- procesi (task) so enote asinhronosti.

Programi v Adi so kolekcije zgoraj opisanih prevajalniških enot. Te lahko shematsko klasificiramo v programske enote takole (ref. 3):



Programske enote so zasnovane tako, da omogočajo, poleg klasičnega strukturiranja v drevesno dosežno hierarhijo tudi poljubno nehierarhično strukturiranje. Odgovarjajo principu skrivanja podatkov (information hiding) z ločitvijo specifikacije programske enote ("vmesniška" (interface) informacija) od implementacije (oz. predstavitve) programske enote (glej npr. jezik Alphard, ref.6), kar omogoča, med drugim, ločeno prevajanje.

Tako zasnovane programske enote omogočajo ustvarjanje knjižnic programskih elementov in varno kombiniranje slednjih v poljubne sisteme (programiranje v "velikem").

3)- paralelno programiranje (zahteve 2,3,4): uvedeni so jezikovni konstrukti za procese kot enote asinhronega izvajanja (procesi so že prisotni npr. v Moduli ali Concurrent Pascalu). Brez procesov si težko zamišljamo programiranje vgnezdenih sprotnih sistemov, kjer so asinhrona dogajanja tipična (o procesih glej ref. 12.).

4)- obravnavanje izrednih situacij (zahteva 1): eksplicitno in sistematično obravnavanje izrednih situacij (exceptions), kot so npr. "overflow", deljenje z 0, indeks polja (array) izven dovoljenega intervala ter poljubnih uporabniško definiranih izrednih situacij je pomemben doprinos zanesljivosti oz. varnosti vgnezdenih sistemov. V Adi je možno izredne situacije definirati, v posebnih konstruktih (exception handler) definirati obravnavanje izredne situacije ter eksplicitno prožiti izredne situacije.

5)- vhodno-izhodne zmožnosti in predstavitvene zmožnosti (zahtevi 3,4). Ada omogoča visokonivojske (datotečni nivo) in nizkonivojske (kmiljenje periferije) vhodno-izhodne zmožnosti. Kombiniranje teh zmožnosti z zmožnostmi, ki jih dajejo operacije za sinhronizacijo in komunikacijo med procesi omogoča realizacijo sprotnostnih zahtev (o tem več v naslednjem poglavju).

Predstavitvene zmožnosti omogočajo specifikacijo predstavitve podatkovnih tipov jezika ter kontroliran dostop do hardwarskih značilnosti računalniškega sistema, kar lahko poveča učinkovitost implementacije ali omogoča programiranje perifernih naprav.

1.2.3. Programi v Adi

Iz doslej povedanega je razvidno, da se način programiranja v Adi precej razlikuje od "standardnega" programiranja.

S programom v Adi opisujemo namreč celoten vgnezden sistem, torej ne samo aktivnosti znotraj računalnikov, ampak tudi interakcije med računalnikom (računalniki) z raznovrstnim hardwarskih okoljem. Program v Adi je poleg tega lahko implementiran na multiračunalniškem sistemu. Druga značilnost Ade (kot sistemsko-aplikacijskega jezika) je ta, da je Adin program edini program, ki se izvaja na procesorjih vgnezdenega sistema, kar pomeni, da izvajalska podpora Ade (tako materialna kot programska) vsebuje vso ustrezno podporo, ki jo na standardnih računalniških sistemih daje operacijski sistem.

Zgornje značilnosti imajo za posledico specifičen pristop k razvoju Adinih programov: ta sloni predvsem na modularnosti (glej ločeno prevajanje, podprogrami, paketi), na močni knjižnični podpori in normalno na prečnem prevajanju: Adini programi bodo verjetno razvijani na razvojnem računalniškem sistemu, ki bo vseboval vse ustrezne udobnosti (Adino programsko podporno okolje: prevajalnik, knjižnice vhodnih in prevedenih prevajalniških enot, ipd.) ter po prečnem prevajanju naloženi v računalnike vgnezdenega sistema.

Vzemimo za primer Adin program za kontrolo postaje za filtriranje rečne vode. Taka postaja vsebuje filtre: voda se pretaka skozi filter in po določenem času filter prečistimo; občasno se lahko pokvarijo zaklopke in je potrebno postajo ustaviti. Adin program za kontrolo postaje bo vseboval zbir ločenih prevajalniških enot. Ena izmed teh bi bila lahko:

```
with MAJOR_PHASES; use MAJOR_PHASES;
  --tu specificiramo vmesniško informacijo:
  --imena drugih paketov, ki jih v tej enoti
  --potrebujemo
procedure SINGLE-FILTER is
  begin
    START_UP;
  loop
    DELIVER_WATER;
    CLEAN_FILTER;
  end loop;
  exception
    when others =>
      --tu opišemo obravnavo izrednih situacij
      --(recimo pokvarjena zaklopka)
    CLOSE_DOWN;
end SINGLE_FILTER;
```

2. PROCESI V ADI

2.1. Opis procesa v Adi

Procesi so v Adi programske enote, ki se lahko izvajajo paralelno. Kadar so procesi implementirani v multiračunalniškem ali multiprocesorskem sistemu, je njihova izvedba lahko dejansko paralelna, kadar pa so implementirali v enem samem procesorju, pa je njihova izvedba navidezno paralelna (izvedbe procesov se v času prepletajo).

Kot bomo videli, procesi pretežno komunicirajo oz. se sinhronizirajo preko procesnih vhodov (entry). Ti vhodi imajo obliko procedur, katerih deklaracija predstavlja vmesniško informacijo o procesu: klic vhodov procesa je edini način direktnega sinhroniziranega komuniciranja s procesom.

Kot vse druge programske enote v Adi tudi proces definiramo v dveh delih: s specifikacijo procesa in s telesom procesa (v tem vrstnem redu). S specifikacijo procesa podamo vmesniško informacijo (interface) o procesu: to v glavnem sestavljajo deklaracije vhodov v proces. Vmesniška informacija je edino, kar uporabnik procesa mora in sme vedeti za komuniciranje s procesom. Telo procesa definira algoritem procesa. Ta delitev procesa na specifikacijski in implementacijski del omogoča, da oba dela nista nujno prevajana skupaj.

V prvem primeru (ref.15) podajamo definicijo procesa v programski enoti, ki bo oče procesa (glej izvajanje 2.2):

Primer 1

```
PARENT;
declare                          --oče procesa ALERT
  P : constant PRESSURE := CURRENT_PRESSURE;
  task ALERT;                          --specifikacija
  task body ALERT is                    --telo procesa
  begin
    if P=LOW then
      NEW_LINE;
      PUT(ASCII.BEL);
    end if;
  end ALERT;
begin              --tu se začne sočasno izvajati
                                        --proces ALERT
--stavki očeta procesa
end PARENT;
```

Iz zgornjega primera je razvidno, da lahko s procesi komuniciramo seveda tudi preko globalnih podatkov (P), vendar mora v tem primeru programer sam poskrbeti za morebitno pravilno sinhronizacijo dosegov do teh podatkov (v zgornjem primeru ni nobenih takih problemov, ker ALERT samo bere podatek P, ki je konstanta).

Drugi primer opisuje proces, ki definira tabelo, ščiteno pred sočasnimi dosegi. Dosegi so realizirani s klici vhodov procesa (READ,WRITE).

Primer 2

```
task PROTECTED_ARRAY is                 --specifikacija
                                        --procesa
  --INDEX in ELEM sta globalna tipa!
  entry READ(N: in INDEX; V: out ELEM);
                --deklaracija vhoda s parametri
  entry WRITE(N: in INDEX; E: in ELEM);
end;                            --konec specifikacije

task body PROTECTED_ARRAY is      --telo procesa
  TABLE : array(INDEX) of ELEM := (INDEX=>0);
                                  --inicializacija
begin
  --algoritem procesa:  neskončna zanka, v ka-
  --teri je selektiran eden od obeh vhodov
  loop
    select
      accept READ(N: in INDEX; V: out ELEM)
      do
                        --prejemni (accept) stavek
                        --specificira "telo" vhoda
        V := TABLE(N);
      end READ;                  --"telo" vhoda READ
    or
      accept WRITE(N: in INDEX; E: in ELEM)
      do
        TABLE(N) := E;
      end WRITE;                 --"telo" vhoda WRITE
    end select;
  end loop;
end PROTECTED_ARRAY;
```

Glede na uporabo bi lahko ločili pasivne in aktivne procese. Kot pasivne bi lahko opredelili tiste procese, ki definirajo mehanizme, potrebne za komunikacijo med aktivnimi procesi ali za sinhronizacijo aktivnih procesov.

Poleg neposredne komunikacije med procesi (preko globalnih podatkov - glej Primer 1), ali po principu rendez-vous-ja, ki je opisan v nadaljevanju), lahko namreč pomensko govorimo o posredni komunikaciji, kjer dva procesa (normalno aktivne narave) komunicirata s pomočjo posredniškega (pasivnega) procesa.

Zgornji primer procesa PROTECTED_ARRAY je dober primer takega posredniškega procesa: definira mehanizem (READ, WRITE v zaščiteno tabelo) preko katerega npr. dva aktivna procesa (prvi kliče procesni vhod WRITE, drugi pa procesni vhod READ) med sabo komunicirata posredno.

Drugi primer posrednega komuniciranja oz. sinhronizacije (o sinhronizaciji procesov govorimo, kadar ne pride do prenosa podatkov med procesi) je pasivni proces tipa SEMAPHORE (Primer 3) ali pa pasivni proces tipa SIGNAL (Primer 4, glej 2.4.2). Procesi tipa SEMAPHORE omogočajo, v primeru, da je semafor uporabljen za zaščito skupnih virov procesov (glej Primer 5) (torej za implementacijo sočasne izključitve) posredno sinhroniziranje procesov, ki kličejo vhoda P in V.

Procesni predmeti (objects) so lahko deklarirani kot procesne konstante (glej PROTECTED_ARRAY), kot spremenljivke tipa proces ali pa so dinamično uvedeni z alokatorjem new, ki kazalcu na procesni tip priredi nov procesni predmet (primerjaj z dinamičnimi spremenljivkami v Pascalu).

Primer 3

```
task type SEMAPHORE is                  --specifikacija
                                        --procesnega tipa
  entry P;        --deklaracija procesnega vhoda
  entry V;
end;

task body SEMAPHORE is
  ACQUIRED : BOOLEAN := FALSE;
begin
  loop
    select
      when not ACQUIRED => accept P;
                        --prejem klica vhoda P je
                        --pogojen z varovalom
      ACQUIRED := TRUE;
    or
      accept V;
      ACQUIRED := FALSE;
    or
      when not ACQUIRED => terminate;
    end select;
```

```
  end loop;
end SEMAPHORE;
  ...

type R is          --vir zaščiten s semaforjem
                   --(glej Primer 5)
  record
    S : SEMAPHORE;          --spremenljivka
                            --procesnega tipa
    D : DATA;
  end record;

type PSEMA is access SEMAPHORE;
                   --tip kazalca na SEMAPHORE
...

PSEMA1 : PSEMA := new SEMAPHORE;
         --kazalec PSEMA1 kaže na nov primerek
         --(predmet) procesnega tipa SEMAPHORE
...
```

Procesni predmeti se obnašajo kot konstante, ker jim ni moč prirejati vrednosti.

2.2. Izvajanje procesov

Preliminarna definicija Ade je vsebovala eksplicitno startanje procesov. Končna definicija (1) je to svobodo skrčila v prid večji strukturiranosti v smislu čistega gnezdenja izvajanj procesov. Procesi (procesni predmeti) so torej aktivirani in njihova telesa začno z izvajanjem takoj po koncu deklaracijskega dela, v katerem je procesni predmet konstantnega procesa ali spremenljivke tipa proces deklariran (glej Primer 1). Procesni predmeti, ki so dinamično alocirani (new), postanejo aktivni takoj po koncu stavka, ki vsebuje alokator new.

Izvajanje procesa se normalno konča, ko procesni algoritem doseže konec procesnega telesa. Če je proces p aktiviral druge procese q(i) (p je oče teh procesov) je p končan šele, ko so končani vsi procesi q(i) (q(i) so sinovi procesa p).

Proces se lahko konča tudi s selekcijo terminate ukaza v selekcijskem stavku (select).

Blok, podprogram ali proces (programska enota), ki je aktiviral proces p je končan šele, ko se konča p. Izvajanje procesov je torej popolnoma gnezdeno.

Procese lahko končamo tudi "nenormalno" - jih abortiramo. O tem kasneje.

2.3. Medprocesorska komunikacija

Kot rečeno, se komunikacija s procesi pretežno in na kontroliran način izvaja preko procesnih vhodov. Procesni vhod je deklariran v specifikaciji procesa v naslednji obliki:

```
entry identifier [(discrete_range)]
                 [formal_part];
```

Formalni del (formal_part) definira parametre vhoda, interval (discrete_range) pa lahko definira družino vhodov.

Vhodi so lahko klicani iz drugih procesov z vhodnim klicem, ki ima obliko procedurinega klica:

```
entry_name [(actual_parameter_part)];
```

Vhodno ime (entry_name) vsebuje še indeks v oklepaju, kadar gre za klic vhoda iz družine vhodov.

Prejemni stavek vhoda, ki je specificiran v procesu p, se lahko pojavi le v telesu procesa p. To pomeni, da lahko proces izvaja samo prejemne stavke svojih vhodov. Za isti vhod lahko proces vsebuje več prejemnih stavkov (istih ali različnih; s tem se vhodi razlikujejo od procedur).

Prejemni stavek ima naslednjo obliko:

```
accept entry_name [formal_part]
[do sequence_of_statements
 end [identifier]
];
```

Vhodno ime (entry_name) vsebuje še indeks v oklepaju, kadar gre za družino vhodov. Zaporedje stavkov med do in end je telo vhoda.

Prejemni stavek lahko specificira telo (med do in end), ki se izvede ob klicu ustreznega vhoda (entry_name). Kadar so parametri (formal_part) in telo prejemnega stavka odsotni, lahko govorimo o sinhronizaciji, ker tedaj ni prenosa podatkov med procesi (glej Primer 3). Formalni del prejemnega stavka mora ustrezati formalnemu delu ustrezne vhodne deklaracije (glej npr. vhod READ in njegov prejemni stavek iz Primera 2).

Komunikacija preko vhodov med procesoma p (kličoči proces, ki kliče vhod V procesa q) in q (klicani proces, ki vsebuje deklaracijo in prejemni stavek (stavke) za vhod V) se izvaja po principu rendez-vous-ja, za katerega je značilno, da je potrebno oba procesa pred samo komunikacijo (interakcijo) sinhronizirati:

- če proces p kliče vhod V procesa q predno je slednji dosegel ustrezni prejemni stavek, je izvedba procesa p suspendirana (dokler q ne doseže prejemnega stavka);
- če proces q doseže prejemni stavek vhoda V predno p kliče ta vhod, je izvedba procesa q suspendirana (dokler p ne kliče vhoda).

Do izvedbe telesa prejemnega stavka v q pride, ko sta oba procesa sinhronizirana: p kliče vhod V in q doseže prejemni stavek vhoda V. Takrat pride do interakcije (rendez-vous), v kateri q izvede prejemni stavek, p pa je suspendiran do konca izvedbe prejemnega stavka. Ob koncu izvedbe prejemnega stavka se oba procesa p in q nadaljujeta.

Kadar več procesov kliče isti vhod V, predno je ustrezni prejemni stavek v klicanem procesu dosežen, so ti klici uvrščeni v vrsto na vhod V (vrsta je fifo). Ob vsaki izvedbi prejemnega stavka za vhod V je najstarejši proces v vrsti na V vzet iz vrste.

Večstranska komunikacija (med več procesi) je omogočena s tem, da telo prejemnega stavka vhoda V lahko vsebuje prejemne stavke (tudi za isti vhod V) ali lahko kliče podprograme, vsebujoče vhodne klice.

Kot bomo videli kasneje, obstajajo tudi pogojni vhodni klici in časovno omejeni vhodni klici.

Prvotni princip rendez-vous-ja je zasnoval Hoare (14) in ima simetrično obliko: vhodni klic navaja klicani proces in prejemni stavek navaja kličoči proces. V Adi je rendez-vous asimetričen (prejemni stavek ne navaja kličočega procesa). Asimetričnost je

utemeljena z naslednjimi razlogi (ref. 2): procese lahko razdelimo na uporabniške (aktivne) in servisne (pasivne) procese; uporabniški procesi morajo poznati servisne procese, ki jih kličejo, klicanim servisnim procesom pa ni nujno potrebno, da poznajo uporabniške procese. Taka asimetrija omogoča formiranje knjižnic servisnih procesov, ki predstavljajo vire za poljubne uporabnike. Ustvarjanje takih servisnih knjižnic pa je ena izmed nujnih zahtev za organizirano (modularno) programiranje vgnezdenih sistemov.

Koncept rendez-vous-ja v Adi združuje komunikacijo in sinhronizacijo procesov in s tem daje enoten mehanizem, uporaben tako za zaščito podatkov kot za signaliziranje med procesi (to bomo videli na primerih). Zaradi tega rendez-vous v Adi predstavlja bistveno novost in prednost pred doslej znanimi mehanizmi, ki so bili namenjeni ali pretežno sinhronizaciji (signali, dogodki, semaforji) ali pa pretežno zaščiti podatkov (semaforji, monitorji, glej ref. 12). Kot je že razvidno iz doslej navedenih primerov, lahko v Adi s procesom in rendez-vous-jem implementiramo tako monitorje (Primer 2: zaščitena tabela z dvema sočasno izključenima dosežnima operacijama READ in WRITE) kot semaforje (Primer 3: proces semaforja z sočasno izključenima dosežnima operacijama P in V).

2.4. Izbirni stavek

Prejemni stavki procesov in klici vhodov omogočajo izražanje dejstva, da proces čaka na dogodek (v prejemnem stavku), ki ga povzroči drug proces (s klicem ustreznega vhoda).

Izbirni stavek (select) dodatno omogoča, da proces čaka na enega izmed večjega števila dogodkov (ta situacija je često prisotna v sprotnih sistemih). Kot bomo videli, ima izbirni stavek izredno izrazno moč. Po obliki je analogen case stavku in v najpreprostejši obliki omogoča izbor enega izmed alternativnih prejemnih stavkov. Značilno zanj je to, da uvaja v izvajanje nedeterminizem (kar pomeni neponovljivost izvajanja), kar je ena izmed karakteristik vgnezdenih sistemov.

Oglejmo si Primer 2: tu imamo izbirni stavek (select) z dvema alternativama: prejem vhodnega klica READ ali prejem vhodnega klica WRITE. Če ob vhodu v izbirni stavek ni bil klican niti vhod READ, niti vhod WRITE, potem v izbirnem stavku proces čaka (je začasno suspendiran) na prvega izmed obeh vhodnih klicev ter nato izvede ustrezno alternativo (med select in or ali or in or ali or in end select). Če je na vhodu v izbirni stavek eden od obeh vhodov že bil klican, je ustrezna alternativa takoj izvedena (z izvedbo alternative je zaključena tudi izvedba izbirnega stavka). Če pa sta na vhodu v izbirni stavek bila klicana že oba vhoda, se na poljuben način (nedeterministično) izbere in izvede ena od obeh alternativ.

S tem nam postane Primer 2 jasen: proces PROTECTED_ARRAY je implementacija ščitene tabele (monitorja), s katero preprečimo sočasno izvajanje doseganja tabele preko vhoda READ in WRITE: kadar se izvaja alternativa, ki vsebuje prejemni stavek za READ, se ne more (sočasno) izvajati druga alternativa, saj sta obe v istem (sekvenčnem) procesu. Klici READ ali WRITE se izvajajo v zaporedju, ki je časovno opredeljeno oz., kadar sta dva različna vhoda klicana "sočasno", je nedeterministično izbran eden izmed obeh.

Ada pozna tri vrste izbirnih stavkov: izbirni čakajoči stavek (selective wait), pogojni vhodni klic (conditional entry call) in časovno omejen vhodni klic (timed entry call).

2.4.1. Izbirni čakajoči stavek

Oblika tega stavka je naslednja:

```
select
    [when condition =>]
        select_alternative
{or [when condition =>]
        select_alternative}
[else
        sequence_of_statements]
end select;
```

"when condition" je varovalo. Zaviti oklepaji označujejo n (≥ 0) kratno ponovitev. Izbirna alternativa (select_alternative) je lahko:

```
accept_statement [sequence_of_statements]
delay simple_expression
                    [sequence_of_statements]
terminate;
```

Odgoditveni stavek (delay) suspendira izvajanje procesa najmanj za čas (merjen v sekundah), izražen z enostavnim izrazom (simple_expression).

Kadar je pred alternativo prisotno varovalo, je alternativa odprta za izbiro, če je ustrezni pogoj res. Alternativa brez varovala je vedno odprta za izbiro. Pomensko bi lahko rekli, da varovalo omogoča onemogočitev (inhibicijo) ustreznih alternativ.

Omejitve: prisotna je lahko ena sama zaključna alternativa (terminate). V istem izbirnem čakajočem stavku ni možno obenem navesti naslednjih alternativ:
- zaključna alternativa, odgoditvena alternativa;
- zaključna alternativa, else -del;
- odgoditvena alternativa, else -del.

Prisotna mora biti vsaj ena prejemna alternativa (alternativa, ki se začne s prejemnim stavkom). Različne prejemne alternative lahko ustrezajo istemu vhodu.

Izvajanje stavka poteka takole:
a) vsi pogoji varoval so izračunani: s tem določimo odprte alternative (alternative, med katerimi lahko izbiramo);
b) prejemna odprta alternativa je lahko takoj izbrana, če je rendez-vous takoj možen (t.j. ustrezni vhod je bil klican);
c) odgoditvena odprta alternativa bo izbrana, če ni v roku, ki ga odgoditev specificira, bila izbrana nobena druga alternativa;
d) zaključna alternativa izraža pripravljenost procesa, da se v tej točki zaključi. Izbrana bo v primeru, da so vsi procesi, ki so bratje in sinovi (posredni ali neposredni) procesa p, ki vsebuje to končno alternativo, končani oz. čakajo v izbirnem čakajočem stavku, ki vsebuje zaključno alternativo in da je oče procesa p (blok, podprogram ali proces, ki je aktiviral p, glej 2.2) končan oz., če je oče proces, da je oče končan ali pa čaka v izbirnem čakajočem stavku, ki vsebuje zaključno alternativo;

e) če nobene alternative ni možno takoj izbrati (vendar vsaj ena odprta alternativa obstaja) in če obstaja else-del, se slednji izvede; če else-dela ni, proces čaka, dokler izbira odprte alternative ne postane možna;
f) če so vse alternative zaprte in če obstaja else-del, je slednji izveden (glej e), sicer pa (če else-del ne obstaja) pride do napake (izredne situacije).

V danem trenutku je lahko možen izbor iz večjega števila odprtih alternativ. V tem primeru je izbor poljuben (nedeterminističen) oz. ga jezik ne definira. Kadar je prisotnih več odprtih odgoditvenih alternativ, sledi iz o), da bo lahko izbrana le tista z najkrajšim rokom.

Odgoditvena alternativa je pomembna v sprotnih sistemih: omogoča, da časovno omejimo čakanje na enega izmed dogodkov, ki ga izražajo prejemne alternative izbirnega stavka (glej Primer 6). Služi lahko za realizacijo tako imenovanih "watch-dog timerjev".
Simetričen mehanizmu časovno omejenega čakanja na dogodek je mehanizem časovno omejenega "pošiljanja" oz. "proženja" dogodka (glej časovno omejen vhodni klic, 2.4.3).

Zaključna alternativa omogoča urejeno zaključevanje procesov: kadar je oče procesa p, ki vsebuje to alternativo končan oz. pripravljen, da konča (čaka v izbirnem čakajočem stavku z odprto zaključno alternativo), kakor tudi bratje in sinovi procesa p, potem se bo ves ta skupek procesov, vključno s p, končal. Glej npr. Primer 3: proces semaforja bomo končali, ko ga nihče več ne potrebuje: oče ga ne rabi več, ker ga ne kliče več (prvi dve prejemni alternativi ni več možno takoj izbrati (b) oz. ker ga ne uporablja več za zaščito podatkov D (v virih tipa R), kar pomeni, da je zaključna alternativa odprta, ker je njeno varovalo res.

Else -del izbirnega stavka predstavlja v bistvu pogojni rendez-vous: če v trenutku izvedbe izbirnega stavka rendez-vous ni takoj možen (glej primera e in f), se izvede else-del. Pogojni rendez-vous ima seveda dve plati: pogojni prejem (v izbirnem čakajočem stavku) in pogojni vhodni klic (glej 2.4.2).

Primer 4

Podajmo primer nememoriziranega signala. Tak signal je lahko poslan (klic vhoda SEND), vendar "pošiljka" ni memorizirana. Tak signal je lahko torej sprejet v procesu p (klic vhoda WAIT) le, če proces p v trenutku, ko je signal poslan, nanj čaka (je suspendiran v klicu vhoda WAIT).

```
task type SIGNAL is
  entry SEND;
  entry WAIT;
end;

task body SIGNAL is
begin
  loop
    select
      accept SEND;
      select          --gnezdeni izbirni stavek:
                      --pogojni rendez-vous
        accept WAIT;
      else
        null;  --če WAITa ni možno takoj spre-
          --jeti, zanko ponovimo, kar pomeni,
          --da je bil signal "izgubljen",
          --ker ni bilo nikogar, ki bi nanj
          --čakal v klicu vhoda WAIT
      end select;
    or
      terminate;
    end select;
  end loop;
end SIGNAL;
```

2.4.2. Pogojni vhodni klic

Pogojni vhodni klic je naslednja oblika izbirnega stavka:

```
select
  entry_call [sequence_of_statements]
else
  sequence_of_statements
end select;
```

Semantika je naslednja: če je mogoč takojšen rendez-vous s klicanim procesom (klicani proces čaka v prejemnem stavku za ustrezni vhod), potem je izvedena alternativa med select in else, sicer pa alternativa med else in end select.

Primer 5

Recimo, da dani vir (vir R iz Primera 3) želimo zaščititi s semaforjem, ki uporablja aktivno čakanje (busy waiting). V ta namen bomo uporabili binarne semaforje iz Primera 3 (proces tipa SEMAPHORE).

```
VIR : R;                              --glej Primer 3

--zaščita z binarnim semaforjem S vira:
VIR.S.P          --klic vhoda P : zasedba vira
--uporaba podatka VIR.D
VIR.S.V        --klic vhoda V : sprostitev vira

--če želimo zaščititi VIR s semaforjem z akti-
--vnim čakanjem, bomo za zasedbo vira uporabi-
--li proceduro PBW:
procedure PBW (SEM: SEMAPHORE) is
  --parameter tipa proces, procedura P z "busy
  --waiting"
begin
  loop
    select
      SEM.P;  --pomen procedure PBW: na klicu
              --vhoda P se vrtimo, dokler ta
              --ni sprejet
      return;          --skok iz neskončne zanke
    else
      null;             --aktivno čakanje, zanko
                        --nadaljujemo
    end select;
  end loop;
end;

--zaščita z aktivnim čakanjem na vir:
PBW (VIR.S);  --zaseg vira z aktivnim čakanjem
--uporaba podatka VIR.D
VIR.S.V                          --sprostitev vira
```

Uporaba zaščite z aktivnim čakanjem se v sprotnih sistemih lahko v določenih okoliščinah pokaže kot učinkovita. Pri binarnem semaforju lahko namreč ob klicu vhoda P pride do čakanja (na rendez-vous s P operacijo), kar povzroči preklop na drug proces. Tedaj čas preklapljanja, čas doseganja vira, ščitenega s semaforjem in čas, ko proces znova poskuša izvesti rendez-vous s P operacijo lahko določajo časovni interval I1, ki je daljši od intervala I2, v katerem je semafor zaseden -

ACQUIRED je res (to je časa, ko se izvajajo operacije nad ščitenim virom). To seveda pomeni, da bi z aktivnim čakanjem (kjer ne pride do procesnega preklopa ob klicu vhoda P) proces lahko hitreje zasegel vir (v času I2 < I1), kar je lahko za sprotnost pomembno. Rešitev z aktivnim čakanjem je lahko primerna za multiprocesorske sisteme, kjer se procesi dejansko odvijajo na različnih procesorjih in navidezno paraleleno izvajanje (prepletanje izvajanj) procesov ni neobhodno.

2.4.3. Časovno omejen vhodni klic

Ta stavek *ima prav tako obliko izbirnega* stavka:

```
select
  entry_call [sequence_of_statements]
or
  delay_statement [sequence_of_statements]
end select;
```

Semantika je naslednja: če je rendez-vous s klicanim procesom lahko izveden v roku, ki ga določa odgoditveni stavek (delay_statement), se izvede sprejem (stavki med select in or), sicer pa se izvede sekvenca stavkov med or in end select.

Tudi ta stavek je pomemben za sprotnost obravnave.

Primer 6

```
task CONTROLLER is
      --gre za nadzornika periferne naprave,
      --ki registrira zahteve treh različnih
      --prioritet za doseg do naprave
  --predpostavimo globalni tip:
  --type LEVEL is (URGENT, MEDIUM, LOW);
  entry REQUEST (LEVEL'FIRST..LEVEL'LAST)
              (D: DATA);  --družina treh vhodov
end;

task body CONTROLLER is
  ...
  select
    accept REQUEST (URGENT)(D: DATA)
    do
            --prejemni stavek za prvi vhod iz
            --družine REQUEST
      ...
    end;
  or
    when (REQUEST(URGENT)'COUNT=0) =>
         --atribut E'COUNT, kjer je E ime vho-
         --da, pove število čakajočih vhodnih
         --klicev v vrsti na E (glej 2.5.2)
    accept REQUEST(MEDIUM)(D: DATA)
    do
      ...
    end;
  or
    when (REQUEST(URGENT)'COUNT=0) and
         (REQUEST(MEDIUM)'COUNT=0) =>
    accept REQUEST(LOW)(D: DATA)
    do
      ...
    end;
  or
    delay 1000.0;          --primer odgoditvene
                           --alternative
    --tu smo, če v 1000 sekundah ni prišlo do
    --nobene zahteve (REQUEST): zbiraj podatke
    --za statistiko pogostnosti zahtev
  end select;
  ...
end CONTROLLER;
  ...
```

```
--v procesu, ki želi uporabljati napravo, za-
--prosimo (podamo zahtevo) za visoko prioritet-
--tni doseg naprave s klicem vhoda REQUEST
--(URGENT) takole:

select              --časovno omejen vhodni klic
  CONTROLLER.REQUEST(URGENT)(SOME_DATA);
or
  delay 45.0;
  --zahteva v 45 sek ni bila sprejeta; očitno
  --je proces CONTROLLER zelo zaposlen: posku-
  --si kaj drugega
end select;
...
```

2.5. Prioritete, atributi in abortiranje procesov

2.5.1. Prioritete

Procesu lahko priredimo statično prioriteto (med izvajanjem procesa se ne spreminja) s pomočjo pragmatskega stavka (pragmat je navodilo prevajalniku) oblike:

```
pragma PRIORITY (static_expression);
```

v specifikaciji procesa. Prioriteta je podana z vrednostjo izraza; nižja vrednost ustreza večji prioriteti; interval prioritet je implementacijsko definiran.

Prioriteto uporablja prevajalnik pri alokaciji procesorskih virov za procese, ki se izvajajo navidezno paralelno (glej 2.1); prioriteta (procesorjev) torej vpliva na razvrščanje procesov.

Za procese z isto prioriteto jezik razvrščanja ne definira.

Prioriteta dveh procesov, ki komunicirata z rendez-vous-jem takole vpliva na prioriteto izvedbe rendez-vous-ja:
- če je prioriteta obeh procesov definirana, se rendez-vous izvede z višjo od obeh prioritet;
- če je definirana prioriteta samo enega procesa, se rendez-vous izvede z najmanj to prioriteto.

Mišljeno je, da naj se te prioritetne specifikacije uporabljajo samo za definiranje relativnih prioritet. Iz tega sledi, da naj jih ne bi uporabljali za realizacijo procesne sinhronizacije (razen morda na nivoju perifernih krmilnikov).

2.5.2. Procesni atributi

Na procesni predmet ali procesni tip T navezujemo naslednje atribute:

T'TERMINATED	: tipa BOOLEAN. (Začetna vrednost je FALSE). Postane TRUE, ko je proces T končan.
T'PRIORITY	: podtipa PRIORITY. Označuje prioriteto procesa, če je ta bila definirana (glej 2.5.1)
T'STORAGE_SIZE	: označuje število pomnilniških enot, alociranih za proces T. Tipa univerzalno celo število.

Procesnemu vhodu E (glej 2.3) procesa T je prirejen naslednji atribut, uporabljiv v telesu procesa T:

E'COUNT : tipa INTEGER. Označuje število vhodnih klicev v čakalni vrsti vhoda E. (Glej primer uporabe v 2.4.3.). Pri uporabi tega atributa moramo biti previdni, ker se njegova vrednost v času izvajanja spreminja.

2.5.3. Abortni stavek

Abortiranje (nenormalno zaključevanje) procesov naj se uporablja samo v zelo resnih primerih.

Abortiranje dosežemo z abortnim stavkom:

```
abort task_name {, task_name};
```

Ta stavek povzroči brezpogojni asinhroni zaključek navedenih procesov, kakor tudi vseh sinov teh procesov. Vsak proces lahko abortira katerikoli proces, vključno sebe.

Če se abortirani proces nahaja v vrsti na procesni vhod, je vzet iz te vrste; če je abortiran kličoči proces sredi rendez-vous-ja, je kličoči proces takoj končan, klicani proces pa normalno zaključi rendez-vous.

Če ima abortirani proces procesne vhode in če obstajajo v vrstah na teh vhodih čakajoči procesi, se te procese ustrezno obvesti (s proženjem izrednih situacij, glej 1.2.2). Če je bil tak čakajoči proces sredi časovno omejenega vhodnega klica (glej 2.4.3), se ustrezni odgoditveni rok izniči.

2.6. Primeri

V nadaljevanju podajamo tri obsežnejše primere programiranja procesov v Adi. Prvi je klasični primer piscev in čitalcev, ki je bil že često opisan v literaturi (služi npr. kot primer uporabe monitorjev, ref. 5), drugi podaja zelo splošen mehanizem za alokacijo skupine virov različnih tipov (ref. 2); tretji pa podaja opis prekinitveno proženega krmilnega procesa za tastaturo (ref. 15).

2.6.1. Pisci in čitalci

Programski enoti, ki definirata kontroliran dostop (pisanje in čitanje) do spremenljivke V sta:

```
package READER_WRITER is
              --specifikacija paketa (vmesniška
              --informacija)
  procedure READ(X: out ELEM);
                --predpostavimo, da je tip ELEM
                --globalno definiran
  procedure WRITE(X: in ELEM);
end;

package body READER_WRITER is     --telo paketa
  V : ELEM := ...;

  task CONTROL is
    entry START;
    entry STOP;
    entry WRITE(X: in ELEM);
  end;

  task body CONTROL is
    RDRS: INTEGER := 0; --število procesov, ki
                        --trenutno čitajo V
  begin
    loop
      select
        when WRITE'COUNT=0 => accept START;
                --če nihče ne čaka za pisanje,
                --dovoli čitanje
        RDRS := RDRS + 1;
      or
        accept STOP;
        RDRS := RDRS - 1;    --zaključi čitalni
                             --proces
      or
        when RDRS=0 =>       --če nihče ne čita,
                             --sprejmi pisanje
        accept WRITE(X: in ELEM)
        do
          V := X;
        end;
        loop          --po pisanju obenem startaj
                      --vse čakajoče čitalce
          select      --pogojni rendez-vous
                      --(prejemna stran)
            accept START; RDRS := RDRS + 1;
          else
            exit;
          end select;
        end loop;
      end select;
    end loop;
  end CONTROL;

  procedure READ(X: out ELEM) is
  begin
    CONTROL.START; X:=V; CONTROL.STOP;
  end READ;

  procedure WRITE(X: in ELEM) is
          --vhod WRITE napišemo kot proceduro,
          --ker vhodov ni možno navesti v
          --specifikaciji paketa
  begin
    CONTROL.WRITE(X);
  end WRITE;
          --tu je konec deklarativnega dela
          --paketa: proces CONTROL je aktiviran
end READER_WRITER;
```

Predpostavljamo, da zgornji paket uporablja skupek uporabniških procesov (aktivnih procesov), ki so ali pisci (kličejo proceduro WRITE) ali pa čitalci (kličejo proceduro READ). Posredniški proces CONTROL (pasivni proces) regulira dostop do deljene spremenljivke V: možno je istočasno čitanje s strani poljubnega števila čitalcev in neistočasno pisanje s strani enega pisca. Ekskluzivno pisanje realiziramo z varovalom na prejemnem stavku za WRITE. Poleg tega je dostop do V-ja reguliran tako, da niti pisci, niti čitalci ne monopolizirajo dostopa do V: varovalo v prejemnem stavku za START dovoljuje začetek čitanja le, če nihče ne čaka na pisanje; po zaključku pisanja so startani vsi čitalci, ki so se medtem nabrali v vrsti na vhod START.

Rešitev ni popolna: čitalci teoretično lahko monopolizirajo dostop do V v zanki za prejemom pisca, če nepretrgoma prihajajo (kličejo vhod START) hitreje, kot jih ta zanka lahko sprejema. Vendar ta slučaj v praksi pomeni, da je sistem preobremenjen in da zahteva drugačno rešitev.

2.6.2. Mehanizem alokacije skupin virov

Primer opisuje paket s kontrolnim procesom, ki omogoča alokacijo skupin virov iz množice virov različnih tipov.

```
package MULTIPLE_RESOURCES is  --specifikacija
                               --paketa
  type RESOURCE is (A,B,C,D,E,F,G,H,I,J,K);
  type RESOURCE_SET is array(A..K) of BOOLEAN;
  procedure RESERVE (GROUP: RESOURCE_SET);
  procedure RELEASE (GROUP: RESOURCE_SET);
end;

package body MULTIPLE_RESOURCES is     --telo
                                       --paketa
  task CONTROL is
    entry FIRST (ASKED: RESOURCE_SET;
                 OK: out BOOLEAN);
                   --prvi poskus alokacije
    entry AGAIN (ASKED: RESOURCE_SET;
                 OK: out BOOLEAN);
                   --ponovni poskus alokacije
    entry RELEASE (GROUP: RESOURCE_SET);
  end;

  task body CONTROL is
    procedure TRY (ASKED:RESOURCE_SET;
                   OK: out BOOLEAN) is
    begin
      if (USED and ASKED) = EMPTY then
        USED := USED or ASKED;
        OK := TRUE;          --uspešna alokacija
      else
        OK := FALSE;
      end if;
    end TRY;
  begin                      --algoritem procesa
    loop
      select
        accept FIRST (ASKED: RESOURCE_SET;
                      OK: out BOOLEAN)
        do
          TRY (ASKED, OK);
        end;
      or
        accept RELEASE(GROUP: RESOURCE_SET)
        do
          USED := USED and not GROUP;
        end;
        for I in 1..AGAIN'COUNT loop
          select
            accept AGAIN (ASKED: RESOURCE_SET;
                          OK: out BOOLEAN)
            do
              TRY (ASKED, OK);
            end;
          else
            exit;            --loop
          end select;
        end loop;            --for loop
      end select;
    end loop;
  end CONTROL;

  procedure RESERVE (GROUP: RESOURCE_SET) is
    POSSIBLE: BOOLEAN;
  begin
    CONTROL.FIRST (GROUP, POSSIBLE);
    while not POSSIBLE loop
      --če prvi poskus ne uspe, poskusi ponovno
      CONTROL.AGAIN (GROUP, POSSIBLE);
    end loop;
  end;

  procedure RELEASE (GROUP: RESOURCE_SET) is
  begin
    CONTROL.RELEASE (GROUP: RESOURCE_SET);
  end RELEASE;

  --proces CONTROL je aktiviran
end MULTIPLE_RESOURCES;
```

Uporabniški procesi uporabljajo paket s klicanjem procedur RESERVE in RELEASE. Procedura RESERVE poskusi takojšnjo alokacijo s klicem vhoda FIRST. Če prvi poskus ne uspe, RESERVE kliče vhod AGAIN, ki pa je lahko sprejet le po dealokaciji virov (po prejemu vhoda RELEASE). Po dealokaciji virov poskuša proces alocirati vire za tiste procese, ki čakajo na vhodu AGAIN (AGAIN'COUNT je izračunan ob vhodu v for zanko). Servisiranje uporabnikov torej ni fifo; odvisno je od razpoložljivih virov.

2.6.3. Krmilnik za tastaturo

Ta primer prikazuje, kako lahko v Adi programiramo nizkonivojske prekinitveno prožene vhodno-izhodne mehanizme ter kako lahko opisujemo hardwarske specifičnosti.

Prekinitev v Adi odgovarja hardwarsko proženemu klicu procesnega vhoda. Ustrezni procesni vhod je opredeljen kot prekinitveni vhod, kadar zanj s posebnim stavkom definiramo fizični naslov prekinitvenega vektorja. Prekinitveno servisno rutino predstavlja prejemni stavek prekinitvenega vhoda. Predpostavlja se, da prekinitveni vhod kliče (hardwarski) proces s prioriteto, višjo od prioritet vseh uporabniških procesov.

Poglejmo si torej krmilnik za tastaturo v PDP-11: prihod znaka iz tastature proži prekinitev na naslovu 100 oktalno, nakar je v servisni rutini znak v podatkovnem registru prepisan v lokalno spremenljivko. Uporabnik uporablja krmilnik preko klica vhoda TAKE, ki prepiše podatkovni register v argument.

```
task KB_HANDLER is
  entry TAKE (CH: out CHARACTER);
  entry KB_DONE;              --prekinitveni vhod
  for KB_DONE use    8#100#;
              --ta specifikacija opredeljuje
              --KB_DONE kot prekinitveni vhod
end KB_HANDLER;

task body KB_HANDLER is
  CHAR: CHARACTER;      --lokalna spremenljivka
  DBR: CHARACTER;       --hardwarski register,
                        --kamor pride znak
  for DBR use 8#177462#; --fizični naslov DBR
begin
  loop                          --zanka krmilnika
    accept KB_DONE              --servisna rutina
    do
      CHAR := DBR;     --prepis znaka v lokalno
                       --spremenljivko
    end KB_DONE;
    accept TAKE (CH: out CHARACTER)
    do
      CH := CHAR;
    end TAKE;
  end loop;
end KB_HANDLER;
```

Jezik ne specificira točne semantike za prekinitvene vhode oz. ustrezne prejemne stavke; to je prepuščeno implementaciji. Tako smo npr. v zgornjem primeru predpostavili, da je prekinitev usposobljena, ko proces doseže accept KB_DONE (s čimer postane klic prekinitvenega vhoda oz. rendez-vous možen) in onesposobljena ob koncu servisne rutine (end KB_DONE). Usposabljanje prekinitve bi bilo možno tudi eksplicitno programirati z doseganjem ustreznega statusnega registra. Zgornji primer prav tako predpostavlja, da proces krmilnika priredi prekinitvenemu vektorju naslov servisne rutine (telo prejemnega stavka za KB_DONE), pri čemer ni čisto jasno, kdaj se to izvede (najprimerneje: ob startu procesa, ko proces prvič doseže accept KB_DONE).

3. ZAKLJUČEK

V tem trenutku o Adi ni možno še ničesar določenega zaključiti. Zgornji pregled definiranja in uporabe procesov v Adi vsekakor kaže, da je Ada bogat in vsestransko uporaben jezik s širokimi zmožnostmi.

Možno je, da se bo uveljavil ne le za programiranje vgnezdenih, temveč tudi neodvisnih računalniških sistemov in sicer za vse nivoje programske opreme (od operacijskega sistema do aplikacij).

Konkretne implementacije in konkretne aplikacije pa bodo pokazale, v kakšni meri je jezik učinkovit in močan oz. v kakšni meri odgovarja zahtevam, ki so mu bile zastavljene.

4. REFERENCE

1. ... Reference manual for the Ada programming language. Proposed standard document, US. Dept. of Defense, july 1980.
2. J. D. Ichbiah et all: Rationale for the design of the Ada programming language, SIGPLAN notices, v. 14, no. 6, june 1979, part B.
3. D. Wegner: Programming with Ada: an introduction by means of graduated examples, SIGPLAN notices, v. 14, no. 12, dec. 1979.
4. W. Eventoff et all: The rendez-vous and monitor concepts: is there an efficiency difference?, SIGPLAN notices, v. 15, no. 11, nov. 1980.
5. J. G. P. Barnes: An overwiev of Ada, Software - practice and experience, v. 10, p. 851-887, 1980.
6. W. A. Wulf et all: Achieving quality software: reflections on the aims and objectives of Alphard. Computer science research review 1975-76, Carnegie-Mellon U.
7. A. K. Jones: The narrowing gap between language systems and operating systems. Ref.: glej 6.
8. ... Mesa language manual, version 5.0, april 1979. Xerox Palo Alto research center, CSL-79-3.
9. M. Exel: Jezik Modula in programiranje sprotnih sistemov, IJS, DP-1771, dec. 1979.
10. B. W. Lampson et all: Report on the programming language Euclid, SIGPLAN notices, v. 12, no. 2, febr. 1977.
11. ... CLU reference manual, Lab. for computer science, M. I. T., LCS/TR-225, oct. 1979.
12. M. Exel: Komunikacija med sekvenčnimi procesi, del I,II, Informatica 1 in 2, 1977.
13. P. Brinch Hansen: Concurrent Pascal Report, Cal. institute of technology report, 1975.
14. C. A. R. Hoare: Communicating sequential processes, Com. of the ACM, v. 21, no. 8, 1978.
15. I. C. Pyle: The Ada programming language, Prentice Hall, 1981.

INFORMATICA 2/1981

ORGANIZACIJA DISTRIBUIRANEGA KRMILJENJA SISTEMA ISKRA 2000

BRANISLAV POPOVIĆ

UDK: 621.395.345:681.326 Iskra 2000

ISKRA TELEKOMUNIKACIJE, KRANJ

V referatu podajamo oris organizacije programske opreme v telekomunikacijskem sistemu ISKRA 2000. Sistem ISKRA 2000 je večprocesorski, modularni in digitalni komutacijski sistem za potrebe javnega PTT omrežja, funkcionalnih omrežij ter za potrebe naročniških central. Krmiljenje sistema je distribuirano s popolnoma decentralizirano podatkovno bazo. V referatu prikazujemo osnovne značilnosti distribuiranega krmiljenja in podajamo pristop k decentralizaciji podatkovne baze v sistemu ISKRA 2000.

ORGANIZATION OF DISTRIBUTED CONTROL IN ISKRA 2000 SYSTEM.
An outline of the ISKRA 2000 telecommunication system software organization in given in the paper. ISKRA 2000 system is multicomputer, modular and digital switching system to be used in public PTT network, dedicated networks and for PABX exchanges. System control is distributed with fully decentralized data base. Basic features of distributed control and an approach to decentralize data base is shown in the paper.

1. UVOD

Sistem ISKRA 2000 je komutacijski sistem telefonskih central za potrebe javnega PTT omrežja /7/, funkcionalnih omrežij /4/ ter naročniških central v podjetjih, hotelih, bolnišnicah ipd./5/.

Sistem je modularen:
Gradi se iz komutacijskih modulov po 128 naročnikov in iz modulov po 30 prenosnikov. Vzdrževalne funkcije so koncentrirane v vzdrževalnem modulu, medtem ko se administrativne funkcije izvajajo v administrativnem modulu. Vsak od modulov se krmili z lastnim mikroračunalnikom. Moduli se povezujejo medseboj preko skupinskega stikala. Skupinsko stikalo komutira tako telekomunikacijske poti kot sporočila pri komuniciranju računalnikov v sistemu.

Sistem je digitalen:
Govor se pretvarja v digitalne PCM vzorce in se v taki obliki vrši tako prenos kot komutacija govora skozi centralo. Zato je komutacijski sistem primeren ne samo za komutiranje govornih poti temveč tudi za komutiranje podatkov iz podatkovnih terminalov. Arhitekturo sistema prikazuje slika 1.

Fizična modularnost sistema zahteva distribuirano krmiljenje telekomunikacijskega prometa: terminali v neki zvezi se krmilijo paralelno z več procesorji - s procesorji, ki krmilijo pripadajoče module teh terminalov. Zato smo pri načrtovanju programske opreme tega sistema, da bi ga čim lažje implementirali in da bi bil čimbolj fleksibilen, izbrali naslednji pristop:

- komutacijske funkcije so popolnoma distribuirane
- podatkovne in administrativne funkcije so koncentrirane v vzdrževalnem oz. administrativnem modulu
- celotno programsko opremo delimo v:
  . sistemsko (operacijski sistem, detekcija napak, start in obnova sistema)
  . aplikativno (obdelava telekomunikacijskih protokolov, administrativni in vzdrževalni postopki)

2. NAČRTOVANJE SISTEMSKE PROGRAMSKE OPREME

Celoten sistem ISKRA 2000 delimo v več hierarhičnih nivojev:
- sistem
- podsistem
- element
- podelement
- enota

Najnižji nivo enote so implementirani programski moduli in procedure, medtem ko so vsi višji nivoji skupek programskih modulov in procedur, ki medseboj sodelujejo, da bi izvedli določeno funkcijo sistema. Programski moduli sodelujejo medseboj preko dobro definiranih vmesnikov.

Pri tako opisanem načrtovanju uporabljamo tehniko SADT (metodologija, ki je bila razvita v Softech) /4, 10/, ki omogoča načrtovanje od zgoraj navzdol in dobro definira vmesnike med sodelujočimi komponentami nekega nivoja. Komponenta na nivoju i je nivo i -1 (npr. komponente podsistema so elementi, komponente pod elementi so enote , ipd.).

Aplikativna programska oprema je podsistem, ki ima za vsako aplikacijo identične vmesnike z ostalimi podsistemi in jo ne načrtujemo po SADT metodi, ker zahteva pri svojem razvoju svojstven pristop.

3. NAČRTOVANJE APLIKATIVNE PROGRAMSKE OPREME

Aplikativna programska oprema sistema ISKRA 2000 so programi, ki obdelujejo:

- telekomunikacijske protokole (signalizacije) med terminalom in centralo ter med centralami v omrežju
- administrativne in vzdrževalne postopke , ki jih zahteva osebje centrale

Osebje centrale izvaja administrativne in vzdrževalne postopke s pomočjo komand v standardiziranem jeziku CCITT - MML , to je jeziku za komunikacijo človeka s strojem (CCITT man-machine language MML) /2/.

Izvršilne rutine teh postopkov so programske procedure z dobro definiranimi vmesniki proti elementu sistema "komunikacija človek-stroj". Če komanda zahteva izvajanje kompleksnejših procesov, potem izvršilna rutina sproži procese, ki se načrtujejo podobno kot obdelava telekomunikacijskih protokolov.

Procesi obdelave telekomunikacijskih protokolov imajo naravo končnih avtomatov, zato le-to aplikativno programsko opremo načrtujemo na specifičen način.

Obdelavo telekomunikacijskega protokola najprej specificiramo z /1/ modificiranim CCITT - SDL diagramom, ki je nekakšna kombinacija diagramov prehajanj stanj avtomata, običajnega diagrama poteka in posebnih konstruktov, značilnih za sistem ISKRA 2000.

Ko je tako specificirana obdelava preverjena s strani poznavalcev telekomunikacijskih signalizacij, napišemo program v SL1 jeziku. SL1 jezik /8, 11/ je tekstualen zapis specifikacije, ki se ga da avtomatsko prevesti v SL1 logične tabele. SL1 tabele določajo, kako se naj telekomunikacijski proces odvija v računalniku sistema.

SL1 tabele interpretira t.im. SL1 interpretor, ki pozivа ustrezne akcije. Akcije, ki se izvajajo ob prehodu strani so enostavne procedure osnovnih operacij nad podatkovno bazo in nad aparaturno opremo.
Akcije SL1 jezika so lahko tudi posebne komande SL1 interpretorju oziroma operacijskemu sistemu.

4. ORGANIZACIJA KOMUTACIJSKIH FUNKCIJ

Obdelavo telekomunikacijske zveze, v kateri imamo dva ali več terminalov, smo razdelili v dve ali več neodvisnih obdelav takoimenovanega polpoziva ali delnih pozivov.
Slika 2 prikazuje konferenčno zvezo treh naročnikov A,B in C. Vsak delni poziv vpliva na drugega (komunicira z drugim) izključno s signali, ki jih imenujemo intertelekomunikacijski signali. Z delitvijo poziva na dva polpoziva ali več delnih pozivov (v primeru konferenčnih zvez) realiziramo funkcije na identičen način tako za intramodularni kot za intermodularni poziv. Intramodularni poziv je poziv, kjer se dva terminala neke zveze nahajata v istem modulu, medtem ko sta pri intermodularnem pozivu terminala zveze v različnih modulih in sta tako krmiljena tudi z različnima procesorjema. Pri tem sta terminala neke zveze lahko priključena na našo centralo z različnima signalizacijama.

Inter-telekomunikacijski signali so standardizirani v sistemu ISKRA 2000 za:

- vse aplikacije in za
- vse signalizacije

Z načelom delnih pozivov in inter-telekomunikacijskih signalov med njimi lahko realiziramo obdelavo signalizacije ene strani, ne glede na to, kakšna je signalizacija na drugi strani.

Intertelekomunikacijski signali se pošiljajo s pomočjo PUT konstrukta v SL1 jeziku preko operacijskega sistema, ki tudi ugotavlja, ali gre za intramodularni ali za intermodularni poziv. Če je poziv intermodularni, operacijski sistem aktivira sistem medprocesorske komunikacije, ki prenese intertelekomunikacijski signal v ponorni modul (slika 3).

Vsak delni poziv delimo dalje na več procesov (slika 4)

- vhodni signalni proces
- obdelava signalizacije (delnega poziva)
- izhodni signalni proces

ter

- skanirni proces

Skanirni proces odkriva spremembe fizičnih signalov na senzorskih točkah perifernih vezij, ki so odraz tokovno napetostnih stanj na linijah k terminalom. Skanirni proces se izvaja periodično s tako periodo T, da lahko zaznamo najhitrejše spremembe fizičnih signalov neke signalizacije. Spremembe v času t odkriva tako, da primerja (aparaturna) stanja na senzorskih točkah v času t s programskim stanjem v času t, ki je slika (aparaturnega) stanja v času t-T. Če se ta stanja razlikujejo, pomeni, da je skanirni proces odkril spremembo in jo pošlje kot ustrezni vhodni signalni primitiv v vhodni signalni proces.

Vhodni signalni proces razpoznava telekomunikacijske signale iz sosledja vhodnih signalnih primitivov. Vhodni signalni proces je torej razpoznavni končni avtomat, ki kot izhod preda telekomunikacijske signale procesu obdelave signalizacije.

Vhodni signalni proces je aperiodičen in ga sproži odkriti vhodni signalni primitiv iz skanirnega procesa.

Obdelava signalizacije je končni avtomat in obdeluje vhodne telekomunikacijske signale iz vhodnega signalnega procesa, izvaja analize sprejetih signalov in podatkov v podatkovni bazi, sproži pošiljanje telekomunikacijskih signalov iz centrale v okolico. Če želi obdelava signalizacije odposlati telekomunikacijski signal, ki je sosledje fizičnih signalov, obdelava signalizacije preda telekomunikacijski signal izhodnemu signalnemu procesu in le-ta oblikuje iz fizičnih signalov želeni telekomunikacijski signal.

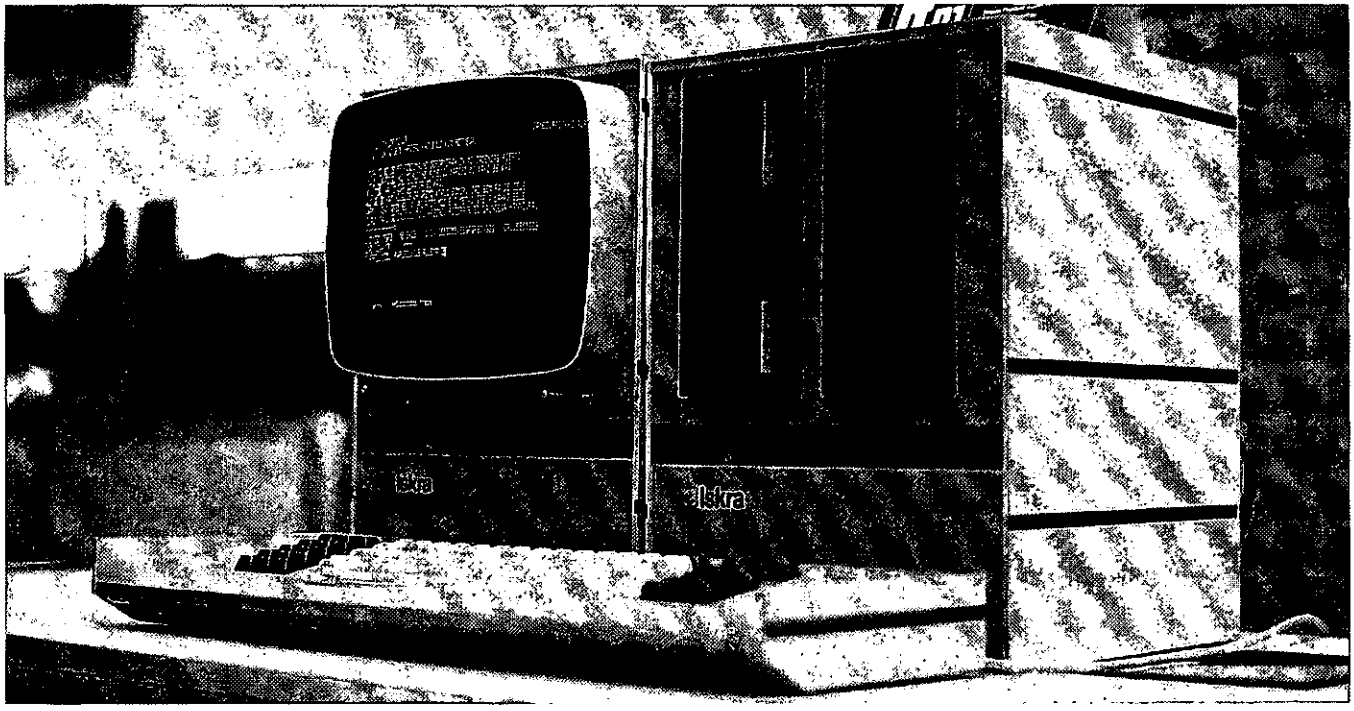
Obdelava signalizacije je aperiodičen proces in ga sproži vhodni telekomunikacijski signal, ki ga je razpoznal vhodni signalni proces.

Izhodni signalni proces je končni avtomat in oblikuje izhodne telekomunikacijske signale, sprejete iz obdelave signalizacije. Izhodne telekomunikacijske signale oblikuje tako, da preko krmilnih točk perifernega vezja generira ustrezno zaporedje fizičnih signalov (ustrezne tokovno-napetostne razmere na liniji k terminalu). Izhodni signalni proces je aperiodičen in

IskraData

Družina računalniških sistemov za distribuirano obdelavo

ISKRADATA 80-50 **Pisalnik**
ISKRADATA 80-60 **Samostojno delovno mesto**
ISKRADATA 80-70 **Komunikacijsko delovno mesto**
ISKRADATA 80-75 **Sinhroni terminal**
ISKRADATA 80-80 **Samostojni matični sistem**
ISKRADATA 80-90 **Komunikacijski matični sistem**



NAJVAŽNEJŠE ZNAČILNOSTI SISTEMA ISKRADATA

- Modulnost sistema
- Prilagodljivost sistema uporabnikovim zahtevam
- Distribuirana obdelava, ki omogoča popolnoma samostojno delo delovnega mesta in dostop do baze podatkov v matičnem računalniku ali nekem drugem računalniku, ki je vključen v omrežje
- 1—16 inteligenčnih delovnih mest
- Disketna enota omogoča direkten pristop do vsakega podatka
- Za večjo količino podatkov se priključijo na matični računalnik do 4 diskovne enote (40, 80, 160, 200, 300 MB)
- Široka paleta možnosti priključitve perifernih naprav (standardni zaporedni in vzporedni vmesnik RS 232-V 24)
- Sinhroni protokoli ali asinhroni protokoli s kontrolo ali brez nje
- Učinkovita programska jezika BASIC in PASCAL
- Delovno mesto je nezahtevno za okolje
- Večja učinkovitost z manjšimi stroški.

Iskra Elektromehanika Kranj TOZD Tovarna računalnikov

informatica 82

Mednarodna razstava računalniške in informacijske tehnologije

Gospodarsko razstavišče Ljubljana
10. do 14. maj 1982

Informatica '82 bo skupna sejemsko-strokovna manifestacija raziskovalnih, proizvajalnih in zastopniških delovnih organizacij, uporabnikov in strokovnih društev s področja računalništva in informatike.

Razstava Informatica je namenjena
- proizvajalcem računalniških naprav in sistemov
- uporabnikom informacijskih sistemov in
- vsem, ki uvajajo automatizacijo z uporabo računalniških sistemov.

RAZSTAVNI PROGRAM

1. Elementi naprav za obdelavo podatkov
- mikroprocesorji
- periferni procesorji
- integrirana vezja in drugi aktivni elementi
- konektorji, kabli, podnožja
- pasivni elementi

2. Enote sistemov za obdelavo podatkov
- centralne enote
- pomnilniške enote (dinamične, statične, mehanične)
- vhodno/izhodno kontrolne enote
- druge podsistemske enote
- usmerniki

3. Periferne naprave in terminali
- enote za čitanje luknjanih kartic
- enote za čitanje in luknjanje traku
- magnetnotračne enote
- enote za pogon magnetnih kaset
- diskovne enote
- pogoni za gibke diske
- optični čitalniki znakov
- čitalniki rokopisov
- znakovni, vrstični in bločni tiskalniki
- video terminali

4. Sistemi za obdelavo podatkov
- sistemi za splošno obdelavo podatkov
- pisarniški sistemi
- razvojni sistemi
- poslovni sistemi
- laboratorijski sistemi
- procesni sistemi
- vojni sistemi
- osebni in domači sistemi
- sistemi za zajemanje podatkov

5. Programska oprema

- osnovna in sistemska programska oprema
- uporabniška programska oprema za poslovne sisteme
- uporabniška programska oprema za vodenje procesov
- komunikacijska programska oprema
- uporabniška programska oprema za tehnične in znanstvene namene

6. Aplikacije računalniških sistemov

- telekomunikacijski sistemi
- vodenje elektroenergetskih sistemov
- zajemanje podatkov
- računalniška grafika
- robotika
- vodenje tehnoloških procesov
- bančni sistemi

7. Naprave za zbiranje in prenos podatkov

- modemi za prenos podatkov
- multiplekserji
- kontrolne in merilne naprave
- naprave za komutacijo

8. Oprema za proizvodnjo računalnikov

- kazalni merilni instrumenti
- elektronski merilni instrumenti
- logični analizatorji
- sistemi za načrtovanje vezij
- sistemi za izdelavo tiskanih vezij
- sistemi za testiranje tiskanih vezij
- sistemi za proizvodnjo integriranih vezij

9. Mediji za zapis podatkov

- magnetni trakovi
- magnetni diski, gibki diski
- magnetne kasete
- formularji na neskončnem traku
- papirni trak

10. Strokovna literatura

- knjige
- revije in časopisi
- uporabniška dokumentacija

Simpozij INFORMATICA 82 je mednarodni simpozij za računalniško tehnologijo in probleme informatike, ki bo v organizaciji Slovenskega društva Informatika, Elektrotehniške zveze Slovenije in Gospodarskega razstavišča v dneh od 10. do 14. maja 1982, obravnaval pereče problematike s področja računalniških znanosti, tehnologije in uporabe. Sodelovanje z mednarodnimi strokovnimi organizacijami bo zagotovilo visoko kakovost referatov in posvetov, kjer bodo sodelovali ugledni mednarodni in domači izvedenci.

Tako bosta razstava in simpozij Informatica '82 srečanje strokovnjakov, proizvajalcev, uporabnikov in drugih interesentov z računalniške informacijske panoge.

INFORMACIJE IN PRIJAVE:
Gospodarsko razstavišče Ljubljana
61000 Ljubljana, poštni predal 113, Litova 50
telefon (061) 311-022, 310-930, 311-232
telex 31 127 gr yu

ga sproži telekomunikacijski signal, ki ga pošlje proces obdelave signalizacije.

Procesi merjenja časov
Našteti procesi potrebujejo pri svojem izvajanju različna merjenja časov, da bi:
- vhodni signalni proces identificiral vhodne telekomunikacijske signale
- izhodni signalni proces generiral izhodne telekomunikacijske signale
- proces obdelave signalizacij časovno nadzoroval prihajanje dogodkov iz okolice

Tako imamo več procesov merjenja časov z različno natančnostjo Δt (= 10 ms, 100 ms, 1 s). Vsakemu terminalu pripada vsakemu od časovnih procesov števec, s katerim odštevamo čas.

Ko želimo v vhodnem signalnem procesu ali v izhodnem signalnem procesu oziroma v procesu obdelave signalizacije izmeriti določen čas, označimo v SL1 programu z "wait" konstruktom vrednost časa in natančnost merjenja. Ko SL1 interpretor odkrije med interpretiranjem SL1 tabel "wait" konstrukt, zaustavi izvajanje tekočega procesa in nastavi vrednost ustreznega števca na želeno vrednost.

Ti časovni števci se odštevajo v časovnih procesih. Ko se števec zmanjša na nič, operacijski sistem ponovno sproži izvajanje procesa od mesta, kjer je bil proces zaustavljen z "wait" konstruktom.

Vsak od procesov obdelave zveze (vhodni signalni proces, proces obdelave signalizacije, izhodni signalni proces) ima svoje časovne procese.

Operacijski sistem komutacijskih modulov
Naloge operacijskega sistema komutacijskih modulov so:
- poganjanje zgoraj naštetih procesov
- interpretiranje SL1 logičnih tabel, s katerimi je določen potek izvajanja aplikativne programske opreme
- prenašanje signalov med procesi
- prenašanje sporočil iz računalnika v računalnik drugega modula preko sistema medprocesorske komunikacije

5. ORGANIZACIJA PODATKOVNE BAZE

Podatki so distribuirani po komutacijskih modulih in načrtujemo sistem tako, da imamo minimalno število skupnih podatkov, da bi dobili maksimalno decentralizirano organizacijo podatkovne baze.

Podatke znotraj komutacijskega modula distribuiramo po terminalih, tako da ima vsak terminal svojo podatkovno bazo v t.im. izravnalniku terminala. Ključ dostopa do vseh podatkov v tem izravnalniku je naslov terminala, medtem ko do posameznih podatkov v izravnalniku lahko pridemo s statističnim odmikom od začetnega naslova izravnalnika.

V splošnem delimo podatkovno bazo na:

- permanetna podatkovna baza (PPB), ki vsebuje:
  - podatke, ki jih programi potrebujejo pri izvajanju določenih operacij (iskanje prvih enic, maske za postavljanje in brisanje bitov, ipd.)
  - SL1 tabele, ki določajo logiko procesiranja pozivov
- semi-permanentna podatkovna baza (SPB), ki vsebuje:
  - podatke o terminalih
  - podatke o telekomunikacijskih smereh
  - podatke o konfiguraciji centrale
  - podatke, ki so potrebni za analizo sprejetih informacij iz okolice centrale
- variabilno podatkovno bazo (VPB), ki jo delimo v :
  - sistemsko variabilno podatkovno bazo (SVPB), ki vsebuje:
    - čakajoče vrste v operacijskem sistemu
    - vmesnike in vlagalnike v operacijskem sistemu
    - podatke v izravnalniku terminala, ki jih uporablja SL1 interpretor pri izvajanju komutacijskih procesov (pri izvajanju končnih avtomatov vhodnega in izhodnega signalnega procesa ter procesa obdelave signalizacij)
    - tabele programskih slik stanj v aparaturni opremi
  - aplikativno variabilno podatkovno bazo (AVPB), ki vsebuje:
    - shranjevalnike rezultatov analiz sprejetih informacij iz okolice centrale

Permanentna podatkovna baza se ne more spreminjati med eksploatacijo sistema. Semi-permanentno podatkovno bazo lahko spreminja samo pooblaščeno osebje centrale s pomočjo komand za komunikacijo človek-stroj preko videoprikazovalnika oziroma preko teleprinterja. Variabilna podatkovna baza se nenehoma spreminja med izvajanjem sistemskih oziroma aplikativnih programov.

Sistemski programi imajo dostop samo do permanentne podatkovne baze PPB in do sistemskih variabilnih podatkov SVPB, medtem ko imajo aplikativni programi dostop do permanentne podatkovne baze PPB, semi-permanentne podatkovne baze SPB ter do aplikativne variabilne podatkovne baze AVPB.

Podatke, ki so skupni celotnemu sistemu in so po svoji naravi podatki, ki naj bi bili v centralni podatkovni bazi, smo organizirali na sledeči način:

- za vse analize podatkov smo iskali možnost distribuirane rešitve ; iskali smo postopke, ki so primerni za distribuirane podatke
- če takega postopka nismo našli oziroma smo prišli z distribuiranim postopkom do nekih skupnih podatkov, smo le-te multiplicirali po modulih

S takim pristopom smo prišli do naslednje organizacije podatkov:

- vsa variabilna podatkovna baza je popolnoma porazdeljena po modulih. Samo stanja sistema medprocesorske komunikacije in stanja digitalnega komutacijskega polja so centralizirana v računalniku skupinskega stikala in so dostopna samo temu računalniku. S tem, da smo vso spremenljivo podatkovno bazo porazdelili v sistemu, smo se izognili možnosti "blokad" procesov, ki so značilna za izvajanje paralelnih procesov v večprocesorskih sistemih.
- vsa permanentna podatkovna baza je multiplicirana po vseh modulih

- naslednji semi-permanentni podatki so multiplicirani po vseh modulih:
  - podatki o porazdelitvi grupe linij naročniške centrale po modulih (samo za javne centrale)
  - podatki o porazdelitvi odhodnih smeri po modulih
  - podatki o porazdelitvi oddajnikov oz. sprejemnikov signalov po modulih
  - podatki za analizo karakterističnih številk

- naslednji semi-permanentni podatki so distribuirani po modulih:

  - vsi podatki o terminalih
  - podatki o porazdelitvi grupe linij naročniške centrale znotraj modula
  - podatki o porazdelitvi smeri znotraj modula
  - podatki o porazdelitvi oddajnikov oz. sprejemnikov signalov znotraj modulov
  - podatki za analizo izbrane katalogne številke

Do zgornje organizacije distribuiranih podatkov smo prišli s tem, da smo določen objekt (smer, grupo linij naročniške centrale ipd.), ki ga želimo distribuirati, najprej distribuirali po modulih. Zato imamo v vseh modulih seznam z navedeno pripadnostjo objekta modulom:

objekt :: ($modul_1$, $modul_J$, ... , $modul_K$)

Znotraj $modula_J$, ki pripada nekemu objektu, na podoben način porazdelimo objekt po terminalih:

objekt (v modulu $_J$) :: ($term._i$, $term._j$...., $term._k$)

Podatki enega terminala se lahko čitajo s procesi drugega terminala v zvezi, kar omogoča GET konstrukt v SL1 jeziku. Ko interpretor izvaja GET konstrukt in je poziv intermodularen, aktivira sistem medprocesorske komunikacije.

6. ORGANIZACIJA ADMINISTRATIVNIH IN VZDRŽEVALNIH FUNKCIJ

Administrativne funkcije se izvajajo v administrativnem modulu. Administrativni modul je sestavljen iz računalnika ter računalniške periferije (teleprinter, video-prikazovalnik, disketne enote). Posadka izvršuje administrativne postopke s posebnim jezikom za komuniciranje človeka s sistemom, standardiziranim s strani CCITT. Človek stroj jezik /2/ omogoča "govoreči" dialog med osebjem in sistemom, da bi osebje lahko spreminjalo podatkovno bazo, ne da bi moralo pri tem poznati detajlno sistem in komande.

Komande interpretira interpretor človek-stroj jezika, ki proži izvajanje izvršilnih rutin administrativnih postopkov. Te izvršilne rutine so v komutacijskih modulih, katerih podatkovno bazo želimo spremeniti.

Vzdrževalne funkcije delimo na:

- sprotno preskušanje in
- občasno (rutinsko) preskušanje sistema

Programi sprotnega preskušanja se izvajajo stalno v sistemu in imajo najnižjo prioriteto. Nahajajo se v komutacijskih modulih in v vzdrževalnem modulu. Programi v vzdrževalnem modulu preskušajo delovanje centralnih aparaturnih enot (medprocesorske komunikacije, komutacijskega polja), medtem ko programi v komutacijskem modulu preskušajo delovanje aparaturnih enot v modulu. Ko programi sprotnega preskušanja odkrijejo nepravilnosti v delovanju modulov ali sistema, obvestijo o napaki vzdrževalni modul, ki na podlagi sprejetih obvestil skuša analizirati in locirati napako ter preda zakodirano obvestilo o napaki in lokaciji napake administrativnemu modulu. Le-ta pa izpiše sporočilo na teleprinter oziroma na video-prikazovalnik v osebju centrale razumljivem jeziku.

Občasno preskušanje sistema in modulov lahko zahteva osebje za vzdrževanje centrale, da bi odkrili napake, ki jih ne odkrivamo s programi sprotnega preskušanja.

Programi za občasno preskušanje se polnijo, ko jih zahtevamo iz disketne enote v pomnilnik vzdrževalnega modula.

Operacijski sistem administrativnega modula
Operacijski sistem administrativnega modula je zbirka sistemskih funkcij (makrojev, subrutin) in je sestavljen iz več plasti:

- jedro (Kernel) /9/, ki razvršča procese in jim dodeljuje resurse - računalniški čas, vhodno/izhodne naprave
- krmilnikov posameznih vhodno/izhodnih naprav
- vhodno/izhodnih funkcij (čitanje / zapisovanje podatkov iz / na periferne naprave)
- datotečnih funkcij (kreiranje, spreminjanje, čitanje datotek na disketi)

7. ZAKLJUČEK

V referatu smo podali osnovne značilnosti programskega sistema ISKRA 2000 in se nismo mogli spustiti v podrobnejši opis nekaterih sicer zanimivih in pomembnih elementov sistema. Celotna zasnova je v fazi implementacije in je že izvedena manjša varianta sistema ISKRA 2000 (brez administrativnega in vzdrževalnega modula), tako da so zasnova distribucije procesiranja komutacijskih funkcij sistema ter zasnova porazdeljene podatkovne baze preverjeni na dejanskem prototipu centrale.

Pri zasnovi in implementaciji tega sistema so sodelovali oziroma sodelujejo številni sodelavci iz Tehničnega razvoja Tovarne avtomatskih telefonskih central in iz odseka za računalništvo in informatiko na Institutu Jožef Stefan.

8. LITERATURA

/1/ CCITT VII th Plenary Assembly, Geneva 1980, Volume VI.7
Functional Specification Description Language (SDL) Recommendations Z.101 - Z. 104

/2/ CCITT VII th Plenary Assembly, Geneva 1980, Volume VI.7
Man-machine language (MML) ; Recommendations Z.311 Z.341

/3/ B.Popović : Povečanje produktivnosti programiranja velikih sistemov
Simpozij Informatica 75, Bled, 1975, 1.33

/4/ B.Popović, M.Dežman : Funkcionalne automatske telefonske mreže sa familijom Iskrinih elektronskih centrala. Zbornik YUTEL 1978, A/5

/5/ A.Mohar, B.Klemenčič, B.Popović : Modular electronic switching system for small and medium size offices, ISS 1979, Paris, Maj 1979

/6/ B.Popović, J.Stare, M.Mekinda: Modularni elektronski komutacioni sistem ISKRA 2000
Zbornik Y UTEL 1979, A/I-1

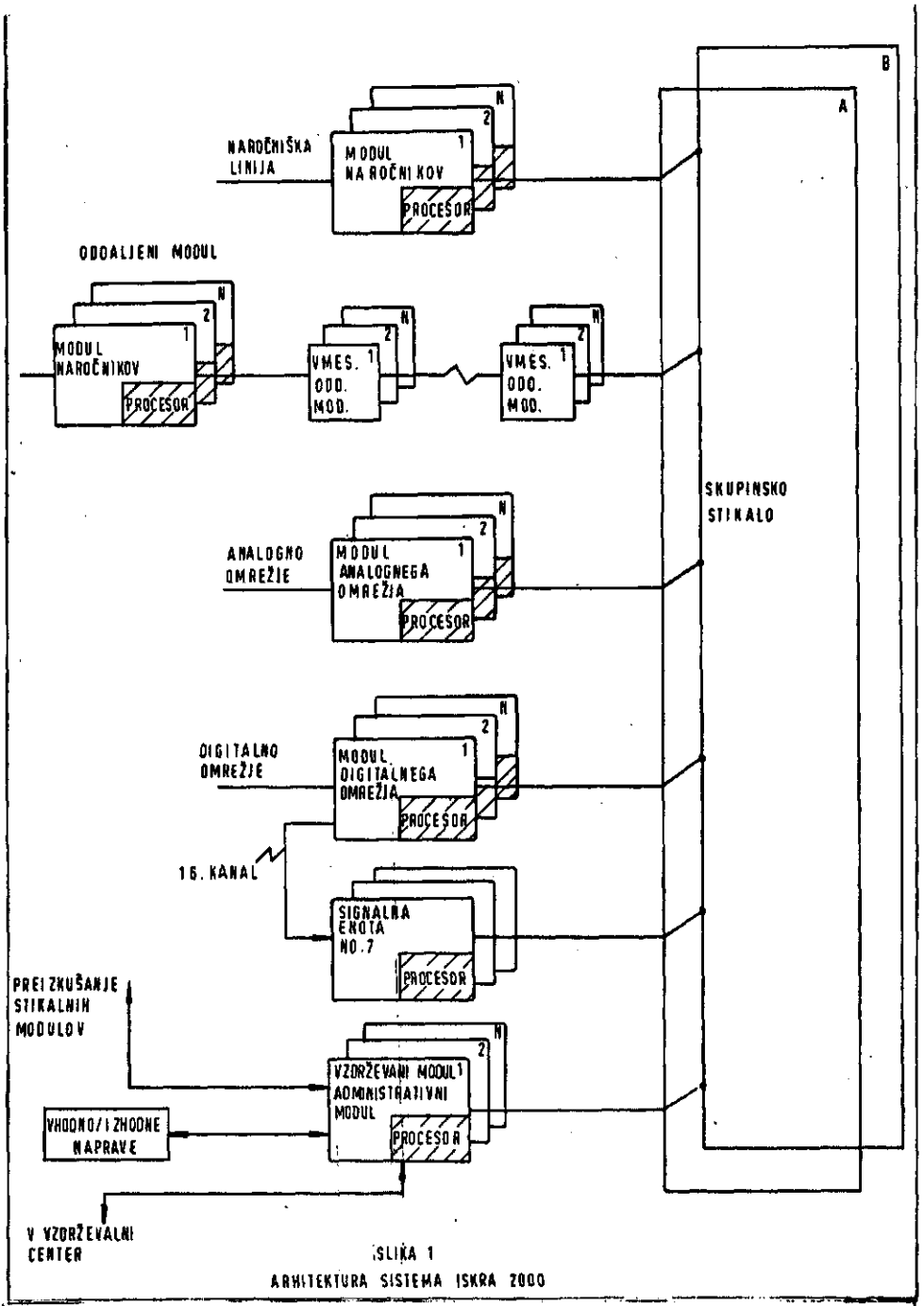
/7/ M.Mekinda, J.Stare, B.Popović: Javna digitalna centrala sa dencetraliziranim upravljanjem

/8/ B.Popović, M.Exel : SL1 jezik in formalni pristop za specificiranje in programiranje komutacijskih procesov.
Zbornik YUTEL 1980, A/III-1

/9/ D.Novak, M.Exel, M.Kovačević, B.Kastelic: Jedro za podporo implementacije sprotnih sistemov.
Časopis Informatica 80, št.2

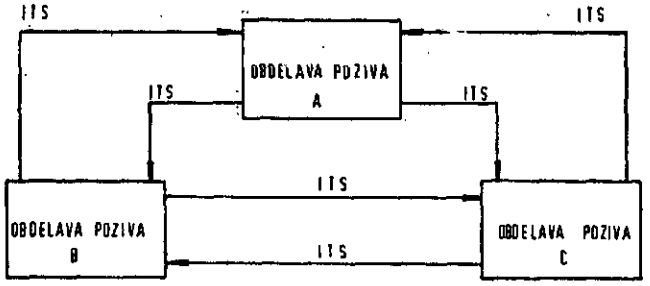
/10/ Structured analysis and design techniques, Softech report, 1974

/11/ M.Exel, B.Popović : SL1 specification language - a tool for switching system software development, ICC 81 , Denver,Colorado, junij 1981



SLIKA 1

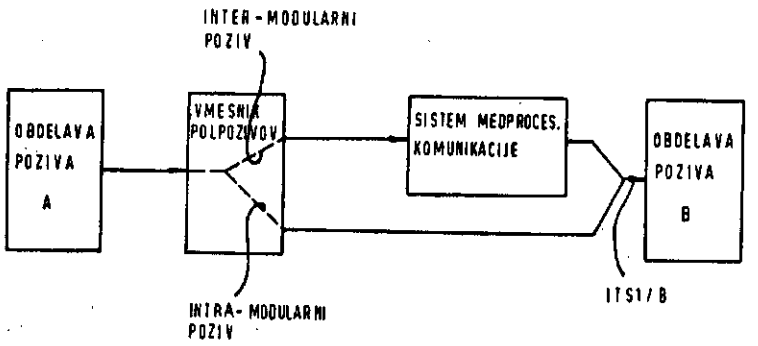
ARHITEKTURA SISTEMA ISKRA 2000



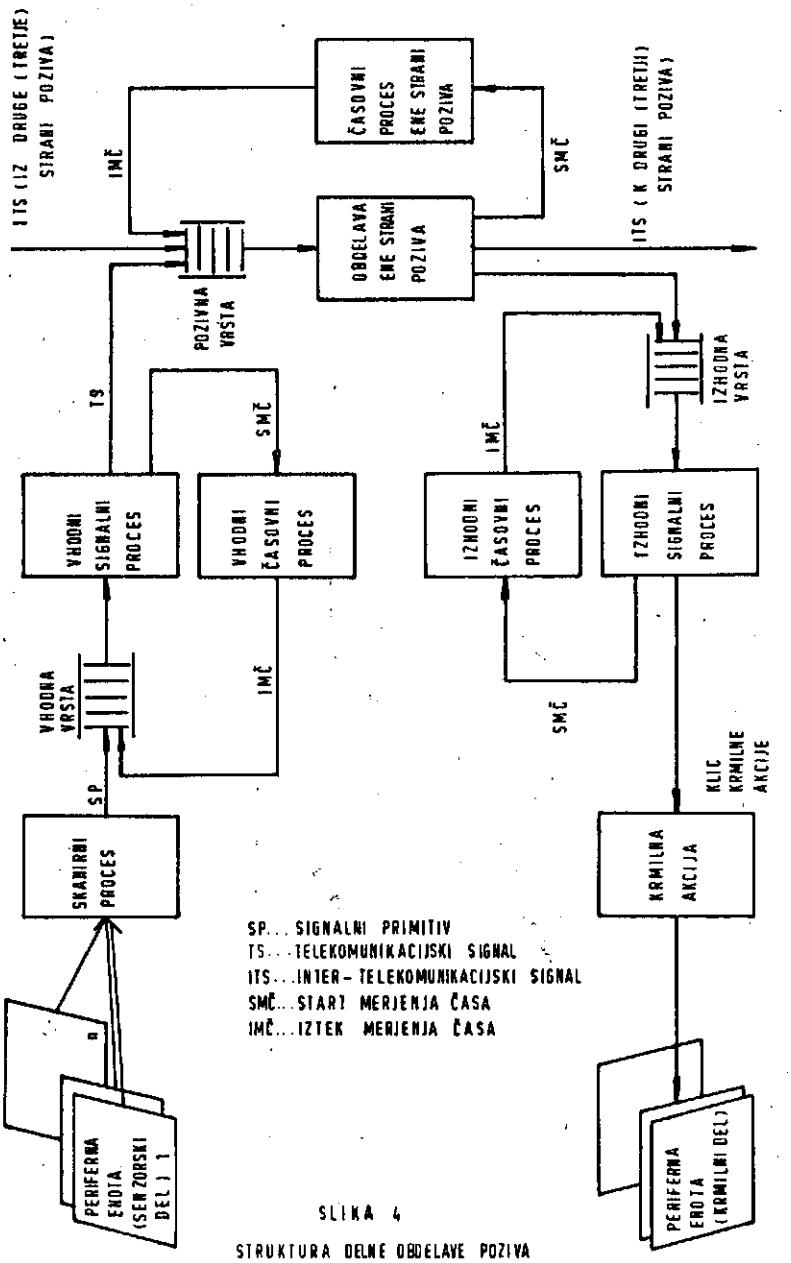
ITS INTER-TELEKOMUNIKACIJSKI SIGNAL

SLIKA 2

PRIMER DELITVE OBDELAVE KONFERENČNE ZVEZE TREH NAROČNIKOV



SLIKA 3
INTER - MODULARNI POZIV



SLIKA 4
STRUKTURA DELNE OBDELAVE POZIVA

INFORMATICA 2/1981

NOVI RAČUNALNIŠKI SISTEM ISKRADATA 19

ANDREJ URATNIK,
LADO PETERNELJ,
JANEZ KOŽUH

UDK: 681.326 Iskvadata 19

ISKRA ELEKTROMEHANIKA
TOZD RAČUNALNIKI KRANJ

Članek je kratek pregled lastnosti novega računalniškega sistema ISKRADATA 19.Sistem je razvit na temelju licenčnega računalnika CDC CYBER 18-20, ki smo ga razširili v skladu s potrebami našega tržišča. Opisane so osnovne lastnosti in možnosti uporabe strojne opreme ter operacijski sistem ITOS 2.1.

THE NEW COMPUTER SYSTEM ISKRADATA 19: The article is a brief overwiev of new sistem ISKRADATA 19 features. Sistem is developed on the basis licensed computer CDC CYBER 18-20. We have extended it according to the needs of our market. Here is the description of the basic features and possibilities of hardware application and ITOS 2.1 operating system.

UVOD

Območje uporabe sistema ISKRADATA C18-20 je pri nas zelo široko: uporabljamo ga v raziskovalnih in tehničnih ustanovah, kakor tudi za poslovne namene. Primeren je v manjših, pa tudi v velikih delovnih organizacijah, v zelo pestrih strojnih in programskih konfiguracijah sistemov. Prav ta vsestranska uporabnost sistemov C18-20 nas je vodila pri odločitvi o razširitvi sistema s stališča strojne, kakor tudi programske opreme in ga zaradi njegovih sposobnosti približati večjim računalniškim sistemom. Zasnovali smo sistem ITOS 2.1, ki je dosežek domačega znanja. Predstavljamo ga vsem, ki se zanimajo za napredek na tem področju.

CILJI PRI RAZVOJU SISTEMA

Pri razvoju sistema ISKRADATA 19 so nas vodili naslednji cilji:

- povečano število uporabniških terminalov - na 32
- hitrejši odzivni čas (višja prioriteta) uporabniškega terminala
- povečano število možnih delovnih postaj (work station) - na 15
- povečana zmogljivost pomnilnika
- sistemski pomnilnik je bilo treba povečati iz obstoječih 64 K na 92 K bytov maksimalno
-celotna zmogljivost pomnilnika (sistemski in uporabniški) je bilo treba povečati od obstoječih 256 k bytov na 512 k bytov.
- pospešno delovanje sistemskih perifernih naprav s poudarkom na vrstičnem tiskalniku

in na programih za posredni izpis
- večja zmogljivost zunanjega pomnilnika (možna uporaba diskovnih enot 40/25 MB, 80/50 MB, 300/180 MB in kombinacij teh)
- uvedba prioritet v paketni obdelavi
- uvedba nekaterih sistemskih storitvenih programov
- organizacija sistema mora biti odprta za nadaljnje razširitve (komunikacije več sistemskih tiskalnikov, krmilnih terminalov z vmesnim pomnilnikom, posebne periferne naprave, itd.) in - večji delež lastne udeležbe pri proizvodnji sistema.

LASTNOSTI PROGRAMSKE OPREME ITOS 2.1

Sistemski disk novega operacijskega sistema vsebuje dva sistema, ki se med seboj razlikujeta glede hitrosti delovanja in števila konfiguriranih ITOS terminalov. Oba sistema imata skupno programsko knjižnico (program library) in sistem datotek (data base in library files-programs).

UPORABNIŠKI A - SISTEM

Ta sistem je enak obstoječemu sistemu ITOS 2.0, le da ima fiksno velikost nezaščitenega področja za paketne obdelave (background) 64 KB. Namenjen je predvsem za prevajanje COBOLSKIH izvornih programov in istočasnemu delovanju ITOS terminalov. V primerjavi s sistemom ITOS 2.0 je mogoče konfigurirati zmogljivost glavnega pomnilnika do 512 KB in efektivno zmogljivost diskovnih enot 25 MB, 50 MB in 180 MB.

UPORABNIŠKI B - SISTEM

Izboljšane lastnosti sistema smo dosegli z naslednjimi popravki in dopolnitvami operacijskega sistema ITOS 2.0:
- reorganizacija uporabe in razširitev pomnilnika, ki ga zaseda operacijski sistem
- povečanje pomnilnika, ki ga zasedajo ITOS terminali
- zamenjava obstoječih krmilnih programov perifernih enot s programi izdaje 1.4 C
- povečanje polja TORPSZ za prenos podatkov na trak ali disketo in obratno od 512 B na 2 KB
- spememba programa za inicializiranje sistema
- dopolnitev programa CONFIG za konfiguriranje sistema in $$POOL datoteke in
- izdelava novih sistemskih UTIL programov.

DELOVANJE KRMILNIH PROGRAMOV

V sistemu ITOS 2.1 so stalno prisotni v pomnilniku naslednji krmilni programi perifernih enot:
- krmilni programi diskovnih enot
- krmilni programi ITOS terminalov
- krmilni programi linijskega tiskalnika
- krmilni programi za posredni izpis na tiskalnikih
- krmilni program disketne enote
- krmilni program simulatorja magnetnega traku na disketni enoti
- krmilni program vhoda paketnih obdelav in
- krmilni program izhoda paketnih obdelav.

Izognili smo se nenehnemu prenosu krmilnih programov iz diskovnih enot v pomnilnik in s tem povečali hitrost delovanja perifernih enot. Hkrati smo modificirali tudi krmilni program za posredni izpis (spooler). V sistemu ITOS 2.0 je bilo delovanje organizirano tako, da je vsak izpisani stavek na tiskalniku zahteval svoje čitanje izpisne datoteke. Sistem ITOS 2.1 pa je organiziran tako, da z enim samim čitanjem izpisne

datoteke dobimo 70-stavkov za izpis na tiskalniku (blocking mode). S tem smo znatno zmanjšali število zahtev za čitanje z diskovne enote in hkrati dosegli, da je hitrost izpisa na vrstičnem tiskalniku enaka njegovi nazivni vrednosti in neodvisna od aktivnosti drugih perifernih enot in obremenjenosti sistema.

VELIKOST POMNILNIKA NAMENJENEGA UPORABNIKOM

Neposredna namestitev krmilnih programov nekaterih perifernih enot v rezidenčnem pomnilniku, blocking - način delovanja programa za posredni izpis, povečanje števila terminalov in povečanje internega polja TOBFSZ, je zahtevalo razširitev sistemskega dela pomnilnika od 64 KB na 92 KB. Večje število ITOS - terminalov uporabnikov zahteva za dobro delovanje sistema (zadovoljiv odzivni čas terminala) povečanje velikosti pomnilnika, namenjenega za uporabnike. Sistem ITOS 2.1 dopušča konfiguriranje velikosti pomnilnika do 512 KB, kar pomeni, da v sistemu z maksimalno konfiguracijo in neaktivnim področjem paketnih obdelav, pripada ITOS uporabnikom 420 KB pomnilnika. Z drugimi besedami, tak sistem dopušča, da je 13 uporabniških programov velikosti 32 KB med izvajanjem stalno nameščenih v pomnilniku (13 x 32 KB = 416 KB). V primeru, ko je zmogljivost pomnilnika manjša od vrednosti, ki jo zahtevajo aktivni ITOS terminali, nastopi izmenjavanje programov iz pomnilnika na diskovno enoto in obratno (swapping). Princip izmenjavanja je zasnovan na enakih principih kot v ITOS 2.0.

KRMILNI PROGRAMI PERIFERNIH ENOT

Sistem ITOS 2.1 vsebuje krmilne programe izdaje 1.4 C, ki so tako kodirani, da so lahko rezidentni v pomnilniku ali diskovni enoti. Tako lahko poljubno, brez vsakršne predelave teh programov, spreminjamo sistem, ki bo imel nekatere programe "core resident", odvisno od načina uporabe sistema in njegovih perifernih naprav. Spremenjeni so tudi nekateri načini prenosa podatkov do perifernih enot (A/Q, ADT, DMA). Tako je na primer krmilni program za vrstični tiskalnik pri izdaji 1.2 C deloval na ADT (automatic data transfer) principu. V izdaji 1.4 C pa deluje v A/Q principu z dvojnim vmesnim pomnilnikom (double buffering).

POVEČANJE POLJA TOBFSZ

Sistem ITOS 2.1 (uporabniški B-sistem) ima velikost polja TOBFSZ 2 KB. To povečanje tudi delno pospeši delovanje celotnega sistema. Po želji uporabnika se lahko to polje poljubno povečuje v mejah, ki jih dopušča področje POOL AREA PARTITION. Da ostane sistem kompatibilen s sistemom ITOS 2.0, smo v uporabniškem A-sistemu ohranili velikost polja 512 KB, kar je posebno pomembno pri prenosu DUMP-informacij iz sistema ITOS 2.1 na sistem ITOS 2.0. V primeru, ko na sistemu ITOS 2.0 čitamo magnetni trak, ki je bil generiran na sistemu ITOS 2.1 (uporabniški B - sistem), sistem izpiše sporočilo.

MOŽNOSTI KONFIGURIRANJA SISTEMA ISKRADATA 19

Spremenjeni program CONFIG in datoteka $$POOL dopuščata naslednje nove možnosti konfiguriranja sistemov:

- do 33 terminalov (MASTER + 32 ITOS USERS)
- do 15 delovnih postaj, ki so lahko dodeljene poljubnemu ITOS terminalu. ITOS terminal z matričnim tiskalnikom predstavlja delovno postajo.
- poljubno kombinacijo do osem diskovnih enot zmogljivosti 25 MB, 50 MB ali 180 MB in

- zmogljivosti pomnilnika od 128 KB do 512 KB v koraku po 32 KB.

NOVI STORITVENI PROGRAMI

Novi sistem ITOS 2.1 vsebuje tudi celo vrsto novih storitvenih programov (UTILITY), ki se lahko uporabljajo tako v A, kakor tudi v B uporabniškem sistemu. Najvažnejši programi so:

- program za čitanje in pisanje IBM-3741, ICOM in MDOS kompatibilnih disket
- uvedba prioritetnega režima delovanja paketnih obdelav (BATCH)
- program za prikaz stanja ITOS sistema in aktivnih uporabniških terminalov.

Prvi program je namenjen predvsem off-line povezavi sistema ISKRADATA 19 s sistemom ID-1680/20 za zajemanje podatkov. Program je uporaben tudi za povezavo s podobnimi napravami drugih proizvajalcev, ki podpirajo format zapisa IBM-3741.

Uvedba prioritetnega režima paketnih obdelav je zasnovana na naslednjih principih:

- vsak program v vrsti za paketno obdelavo dobi začetno prioriteto, ki je odvisna od številke terminala in efektivne velikosti programa (vsak terminal ima vnaprej predpisano prioriteto)
- prioriteta vseh čakajočih programov s časom linearno narašča
- vsi parametri prioritetnega načina obdelav (začetne vrednosti prioritet za posamezne terminale, hitrost naraščanja) so pod neposredno kontrolo operaterja
- operater lahko dodeljuje absolutno prioriteto posameznim čakajočim programom, lahko pa tudi uvede stari FIFO (first in/first out) način obdelav.

Program ITOS/USERS STATUS daje operaterju popoln vpogled v dogajanje v sistemu. Program daje sliko vseh aktivnih ITOS terminalov, ime lastnika, procedure ter ime in velikost programa, ki se trenutno izvaja na določenem terminalu. Hkrati daje informacije o zasedenosti pomnilnika s strani operacijskega sistema in ITOS terminalov, velikost in dogajanje v nezaščitenem področju, odstotek zasedenosti pomnilnika ter druge sistemske informacije. Te informacije so zlasti koristne pri sistemih z oddaljenimi terminali. Tako ima operater popoln nadzor nad dogajanjem v sistemu, kar je za interaktivni sistem zelo koristna informacija.

NOVI PROGRAMSKI JEZIKI NA SISTEMU ITOS 2.1

Na sistemu ITOS 2.1 se lahko uporabljajo naslednji novi prevajalniki:

- križni zbirnik CA6800 za prevajanje izvornih programov procesorja M6800. Program generira izhodno listo prevedenega programa z referenčno listo spremenljivk in datoteko prevedenih strojnih kod
- križni zbirnik CA68000. Program je podoben programu CA6800, le da je namenjen za prevajanje izvornih programov procesorja M68000.
- RATFOR - fortranski predprevajalnik. Programski jezik sam omogoča strukturirano programiranje in s tem razumljivejše in lažje pisanje programov.

LASTNOSTI STROJNE OPREME SISTEMA ISKRADATA 19

V sistemu ISKRADATA 19 smo glede na vzajemno delovanje strojne in programske opreme in glede na željene lastnosti celotnega sistema vpeljali naslednje bistvene spremembe v strojni opremi:

Razširitev pomnilnika z 256 KB na 512 KB z dodatno vgraditvijo štirih pomnilniških

plošč. Pomnilnik lahko tako širimo po 64KB pomnilniških modulih.

Razširitev podatkovnih kanalov (A/Q in AQ-DMA kanala) za povečanje števila vhodno-izhodnih enot. V ta namen smo razvili posebno vmesno vezje AQ EXT , ki nam omogoča vgraditev po 5 krmilnikov za prenos podatkov prek kanala AQ in 5 krmilnikov za prenos preko AQ-DMA kanala. Tako bo lahko na procesor priključenih do 32 terminalov preko štirih 8 kanalnih adapterjev, do dva vrstična tiskalnika preko dveh kontrolerjev za tiskalnik in nestandardne periferne naprave (na primer grafični terminal, drugi računalniki itd.).

Da bi povečali hitrost prenosa podatkov med procesorjem in terminalom, uvajamo nov tip krmilnika terminalov (BCLA) z vmesnim pomnilnikom, ki omogoča DMA prenos podatkov v centralni pomnilnik.

Razvili smo tudi vmesnik po standardu RS 422, ki omogoča priključitev terminalov na procesor s kabli dolžine do 1200 m. Vmesnik je izpeljan tako, da na standardni 8 kanalni adapter priključimo posebno vezje (standard RS 422). Medtem ko ima naš terminal ID 168 možnost priključitve na ta vmesnik standardno, je v primeru uporabe terminala CDC 752 potrebno le-tega modificirati.

Razvita je tudi prilagoditev krmilnika diskovnih enot, tako da bo mogoče na procesor priključiti tudi diskovne enote z zmogljivosti 180 MB, z možnostjo katerekoli kombinacije med diskovnimi enotami: 25 MB, 50 MB in 180 MB do skupno 8 enot.

Zaradi razširitev in izboljšav obstoječega sistema ISKRADATA 18-20 ter zaradi možnosti dodajanja posebnih perifernih naprav, smo v sistemu ISKRADATA 19 predvideli dodaten prostor za 26 modulov, kot so pomnilniške plošče, dodatni krmilniki ali posebni krmilniki. Zahteve po večjem številu modulov smo izpolnili tako, da smo zasnovali kabinet z dvema ohišjema modulov. Kabinet v obliki omare izmer 1610 x 750 x 750 mm vsebuje poleg navedenih dveh okrovov modulov še ustrezne napajalnike, vezje za vklop in distribucijo električne energije ter eno ali dve disketni enoti.

Izboljšava aparaturne opreme je zahtevala tudi razvoj novih diagnostičnih programov, tako da je potekal istočasno z razvojem strojne opreme tudi razvoj diagnostičnih programov za AQ, AQ-DMA razširitve in BCLA krmilnik. V diagnostične programe uvajamo tudi dodatna sporočila o napakah v sistemu.

PREHOD SEDANJIH UPORABNIKOV NA NOVI OPERACIJSKI SISTEM

Pri izdelavi novega sistema je bilo tudi poskrbljeno za enostaven prehod uporabnikov iz starega na novi operacijski sistem. Tako mora uporabnik spoznati le nekatere nove sistemske funkcije, obstoječih aplikativnih programov ni potrebno popravljati oziroma ponovno prevajati. Sistem je na razpolago v dveh osnovnih oblikah:

- ITOS 2.1 SMD SL-156 - operacijski sistem na SMD diskovni enoti (Storage Module Drive) izdaja SL-156
- ITOS 2.1 SMD SL-156 COMM - operacijski sistem na SMD diskovni enoti in komunikacijskim podsistemom COMM izdaja SL-156.

Oba sistema se lahko tudi brez omejitev instalirata na obstoječo strojno opremo, le da je sistem ustrezno konfiguriran.

POVZETEK

Novi računalniški sistem ISKRADATA 19 je bil zgrajen na temelju licenčnega računalnika CDC CYBER 18-20, ki smo ga razširili v skladu

s potrebami našega tržišča. V razvojnem oddelku naše tovarne smo za dalj časa angažirali večjo skupino strokovnjakov in ti so razvili in pripravili kompletno proizvodno dokumentacijo za sistem ISKRADATA 19, ki ga s ponosom in upravičeno lahko imenujemo "naš računalnik". Serijsko ga bomo dobavljali v letu 1982, sedanji uporabniki C 18 pa bodo imeli možnost, da uvedejo novi operacijski sistem ITOS 2.1 že v drugi polovici leta 1981. Novi računalnik bo nedvomno lahko v polni meri zadovoljil naše sedanje in bodoče uporabnike. Omogočil bo priključitev večjega števila terminalov, boljše delovanje perifernih enot in hkrati povezovanje v mreže za prenos podatkov.

INFORMATICA 2/1981

JEZIK PL/I IN MIKRORAČUNALNIKI III

ANTON P. ŽELEZNIKAR

UDK: 681.3.06 PL1:181.4

SOZD ELEKTROTEHNA, DO DELTA

Članek opisuje v svojem tretjem delu nekatere posebne uporabe programirnega jezika PL/I-80. Ta del članka začenja s prevajanjem po delih (modulih) in s povezavo delov, ko obravnava primere s pridevkoma EXTERNAL in ENTRY. Primer ločenega prevajanja in kasnejše povezave z LINK ukazom je posebej in podrobno obdelan. Obdelava znakovnih nizov z uporabo vgrajenih funkcij obravnava primer preprostega prevajanja vhodnih stavkov naravnega jezika. Posebno poglavje je namenjeno obdelavi zbirk, kjer je obdelan primer kopiranja zbirk ter programa za oblikovanje in razpoznavanje zbirk imen in naslovov. Zadnje poglavje obravnava okvirno še uporabo PICTURE formata.

PL/I Language and Microcomputers III. This article deals with some particular application of PL/I-80. The third part of the article begins with problems and examples of separate compilation and linkage of program units (modules) using EXTERNAL data items and ENTRY attribute. An example of separate compilation and linkage using LINK command is presented in detail. A simple example of translation of natural language input sentences is shown by usage of built-in string functions. Further, the file processing is presented by programs for file-to-file copying, and for file creating and recognising. The last chapter of the article deals with the PICTURE format usage.

9. Uvod k tretjemu delu

V prvih dveh delih članka smo opisali zgradbo jezika PL/I-80 in v drugem delu smo si ogledali prve primere programov v tem jeziku. Tretji del članka je namenjen izključno uporabi jezika PL/I-80 in v njem bomo obravnavali nadaljne konkretne primere. Posebej si bomo ogledali možnosti uporabe PICTURE formata, ki je zanimiv za reševanje nekaterih poslovnih nalog.

Programiranje v jeziku PL/I zahteva več izkušenj, več znanja o uporabi jezikovnih pripomočkov kot npr. programiranje v jeziku tipa BASIC. Ta občutek si je moč pridobiti že v enostavnih primerih, zlasti tistih, ki uporabljajo V/I stavke za urejanje zbirk. Seveda pa je moč v vrsti primerov oziroma stavkov enega in drugih jezikov potegniti vzporednice: največkrat gre le za razlike v rezerviranih besedah, ki so namenjene podobnim funkcijam.

10. Prevajanje po delih in povezava delov

Programiranje je lažje, če razbijemo velike (dolge) programe v module, ki jih kasneje povežemo med seboj ter s podprogramsko knjižnico. Za ločeno prevajanje in povezovanje imamo dva bistvena vzroka: dolgi programi se dalj časa prevajajo, obstaja pa tudi večja verjetnost prekoračitve obsega pomnilnika, ki je namenjen za simbolno tabelo. Majhne programske segmente razvijemo neodvisno, jih preizkusimo in povežemo: njihovo prevajanje je tudi enostavnejše. Programer bo dokaj hitro opazil, da obstajajo posebne subrutine, ki so se kako priročne pri njegovem uporabniškem programiranju. Iz takih subrutin se lahko oblikuje uporabniška knjižnica, ki vsebuje subrutine za povezovanje v različne nove programe. Oglejmo si kratko povezovanje programskih in podatkovnih segmentov s primeri ločenega prevajanja in povezovanja.

10.1. Podatkovna in programska določila

Podatkovna območja je moč deliti (med več segmenti) z uporabo EXTERNAL pridevka v določilu (deklaraciji) segmenta. Npr. določilo

```
DCL w (10) FIXED BINARY EXTERNAL;
```

določa spremenljivko w, ki zaseda 10 fiksnih binarnih lokacij (20 sosednjih zlogov). Te lokacije so dostopne iz vseh tistih drugih modulov, ki uporabljajo enako določilo. Tudi določilo

```
DCL 1 x,
      2 y (10) BIT (8),
      2 z CHAR (9) VAR;
```

določa 20 zložno podatkovno območje x, ki je dostopno iz drugih modulov. Za določila t.im. zunanjih podatkov velja tole:

```
A_TYPE POZIV.PRN

PL/I-80 V1.3  COMPILATION OF: POZIV

D: DISK PRINT
L: LIST SOURCE PROGRAM

   NO ERROR(S) IN PASS 1

   NO ERROR(S) IN PASS 2

PL/I-80 V1.3  COMPILATION OF: POZIV

   1 A 0000 POZIV:
   2 A 0006      PROC OPTIONS(MAIN);
   3 C 0006      DCL
   4 C 0006          F (3) ENTRY (FLOAT) RETURNS (FLOAT) VARIABLE,
   5 C 0006          G ENTRY (FLOAT) RETURNS (FLOAT);
   6 C 0006      DCL
   7 C 0006          I FIXED, X FLOAT;
   8 C 0006
   9 C 0006      F(1) = SIN;
  10 C 000C      F(2) = G;
  11 C 0015      F(3) = H;
  12 C 001E
  13 C 001E          DO I = 1 TO 3;
  14 C 0030          PUT SKIP LIST('VSTAVI X ');
  15 C 004C          GET LIST(X);
  16 C 0067          PUT LIST('F(',I,')=',F(I)(X));
  17 C 00BD          END;
  18 C 00BD      STOP;
  19 C 00C0
  20 C 00C0      H:
  21 C 00C0          PROC(X) RETURNS (FLOAT);
  22 E 00C0          DCL X FLOAT;
  23 E 0007          RETURN (2*X + 1);
  24 C 00DB          END H;
  25 A 00DB      END POZIV;

CODE SIZE = 00DB
DATA AREA = 0025
FREE SYMS = 0296
END  COMPILATION
```

Lista 8. Program POZIV na levi kaže uporabo ENTRY spremenljivk in ENTRY konstant. Zanimivo je tudi oblikovanje funkcije F(I)(X) v vrstici 16, ki z indeksnim mehanizmom doseže predstavitev treh različnih funkcij (sin, G in H). Funkcija G je zunanja ter je lahko definirana v nekem drugem modulu, tako da se ločeno prevede in kasneje poveže v programsko celoto. Povezava ločenih prevedenih modulov se opravi z LINK ukazom. Kasneje si bomo ogledali konkreten primer z uporabo LINK ukaza.

Programska lista 8 kaže primere ENTRY konstant in ENTRY spremenljivk. Ta program ima 4 vstopne konstante in sicer: glavni program z označitvijo POZIV, zunanjo proceduro G, določeno z vrstico 5, funkcijo SIN iz PL/I knjižnice ter notranjo funkcijo H, ki začenja z vrstico 20. Kot ena ENTRY spremenljivka je v vrstici 4 določena F s tremi elementi (indeksi). Posamezni indeksni elementi za F se inicializirajo v vrsticah 9 do 11 s konstantami SIN, G in H (SIN je funkcija iz knjižnice, G je ENTRY (zunanji)element, H je procedura).

DO skupina (vrstice 13 do 17) pripravi konzolo za oddajo vsake od teh funkcij (vrstica 16), ko se funkcije pokličejo z obliko

F (I) (X)

tj. z

SIN X, G (X) in H (X)

Prvi par oklepajev določa indeks, drugi par pa dejanski parameter.

- zunanjemu podatku se avtomatično priredi STATIC pridevek;
- EXTERNAL podatki so dostopni v vsakem bloku, kjer so deklarirani in tako prekrivajo območje za notranje podatke;
- EXTERNAL podatki se morajo ujemati v prvih šestih znakih, ker povezovalno urejevalni format zahteva zaokroženje od sedmega znaka naprej;
- območja EXTERNAL podatkov morajo biti določena z enako dolžino v vseh modulih, v katerih se ta določila pojavljajo;
- izogibljemo se uporabi znaka "?" v imenih spremenljivk, ker se ta znak uporablja kot prefiks pri knjižničnih imenih;
- samo v enem modulu je dopustna inicializacija EXTERNAL podatka, ki se navaja v več modulih.

ENTRY konstante in ENTRY spremenljivke so podatki, ki rabijo tudi za identifikacijo procedurnih imen in vrednosti procedurnih parametrov. ENTRY konstante ustrezajo proceduram, ki so določene v okviru programa (notranje procedure) ali v času povezovanja (zunanje procedure). ENTRY spremenljivke dobijo vrednosti ENTRY konstant med izvajanjem programa z direktnim prireditvenim stavkom ali pa s prireditvijo dejanskega parametra formalnemu parametru v subrutinskem pozivu. Proceduro lahko aktiviramo s pozivom na ENTRY konstanto ali posredno s pozivom vrednosti procedurne konstante, ki jo hrani ENTRY spremenljivka. ENTRY spremenljivka je lahko indeksirana.

10.2. Primer ločenega prevajanja

Oglejmo si primer ločenega prevajanja in povezovalnega urejanja. Programa na listah 9 in 10 oblikujeta modul, ki v sodelovanju s konzolo rešuje sistem hkratnih enačb.

Program na listi 9 deluje s pomočjo konzole, ko včitava koeficiente in desne strani sistema enačb. Program na listi 10 oblikuje inverzno matriko za rešitev sistema enačb. Pri tem je procedura INV (lista 9) glavni program (vsebuje OPTIONS (MAIN)), dočim je program INVERT (lista 10) subrutina, ki jo pokliče glavni program. V vrstici 16 liste 9 se nahaja določilo ENTRY konstante INVERT, ki se v glavnem programu pokliče z vrstico 46. Parametri INVERT subrutine so določeni v vrstici 18 kot števila s plavajočo vejico matrike (MAXROW krat MAXCOL).

INVERT procedura v listi 10 ima formalne parametre A, R, C, kot je določeno v vrstici 2 in deklarirano v vrsticah 7 in 8. Znaki + v posameznih vrsticah (za številko) se pojavijo zaradi INCLUDE zbirke, iz katere se vzameta MAXROW in MAXCOL.

Povezava obeh programskih modulov (točneje zbirk) se opravi s PLI direktivo

LINK INVERT.COM = INVERT1, INVERT2

kjer je INVERT1 ime zbirke s programom INV ter INVERT2

```
B_TYPE INVERT1.PRN

PL/I-80 V1.3  COMPILATION OF: INVERT1

D: DISK PRINT
L: LIST SOURCE PROGRAM

%INCLUDE 'MATSIZE.LIB';
   NO ERROR(S) IN PASS 1

   NO ERROR(S) IN PASS 2


PL/I-80 V1.3  COMPILATION OF: INVERT1

   1 A 0000 INV:
   2 A 0006      PROCEDURE OPTIONS(MAIN);
   3 C 0006      %REPLACE
   4 C 0006          TRUE   BY '1'B,
   5 C 0006          FALSE  BY '0'B;
  6+C 0006       %REPLACE
  7+C 0006           MAXROW BY 26,
  8+C 0006           MAXCOL BY 40;
   9 C 0006      DCL
  10 C 0006          MAT(MAXROW,MAXCOL) FLOAT (24);
  11 C 0006      DCL
  12 C 0006          (I,J,N,M) FIXED(6);
  13 C 0006      DCL
  14 C 0006          VAR CHAR(26) STATIC INITIAL
  15 C 0006          ('ABCDEFGHIJKLMNOPQRSTUVWXYZ');
  16 C 0006      DCL
  17 C 0006          INVERT ENTRY
  18 C 0006              ((MAXROW,MAXCOL) FLOAT(24), FIXED(6), FIXED(6));
  19 C 0006
  20 C 0006      PUT LIST('SOLUTION OF SIMULTANEOUS EQUATIONS');
  21 C 001D          DO WHILE(TRUE);
  22 C 001D          PUT SKIP(2) LIST('TYPE ROWS, COLUMNS: ');
  23 C 0039          GET LIST(N);
  24 C 0052          IF N = 0 THEN
  25 C 0059              STOP;
  26 C 005C
  27 C 005C          GET LIST(M);
  28 C 0075          IF N > MAXROW | M > MAXCOL THEN
  29 C 0087              PUT SKIP LIST('MATRIX IS TOO LARGE');
  30 C 00A6          ELSE
  31 C 00A6              DO;
  32 C 00A6              PUT SKIP LIST('TYPE MATRIX OF COEFFICIENTS');
  33 C 00C2              PUT SKIP;
  34 C 00D3                  DO I = 1 TO N;
  35 C 00E8                  PUT LIST('ROW',I,':');
  36 C 0119                  GET LIST((MAT(I,J) DO J = 1 TO N));
  37 C 0173                  END;
  38 C 0173
  39 C 0173              PUT SKIP LIST('TYPE SOLUTION VECTORS');
  40 C 018F              PUT SKIP;
  41 C 01A0                  DO J = N + 1 TO M;
  42 C 01B7                  PUT LIST('VARIABLE',SUBSTR(VAR,J-N,1),':');
  43 C 01F3                  GET LIST((MAT(I,J) DO I = 1 TO N));
  44 C 024D                  END;
  45 C 024D
  46 C 024D              CALL INVERT(MAT,N,M);
  47 C 0253              PUT SKIP(2) LIST('SOLUTIONS:');
  48 C 026F                  DO I = 1 TO N;
  49 C 0284                  PUT SKIP LIST(SUBSTR(VAR,I,1),'=');
  50 C 02B4                  PUT EDIT((MAT(I,J) DO J = 1 TO M-N))
  51 C 0314                      (F(8,2));
  52 C 0314                  END;
  53 C 0314
  54 C 0314              PUT SKIP(2) LIST('INVERSE MATRIX IS');
  55 C 0330                  DO I = 1 TO N;
  56 C 0345                  PUT SKIP EDIT
  57 C 03AF                      ((MAT(I,J) DO J = M-N+1 TO M))
  58 C 03AF                      (X(3),6F(8,2),SKIP);
  59 C 03AF                  END;
  60 C 03AF              END;
  61 C 03AF          END;
  62 A 03AF END INV;

CODE SIZE = 03AF
DATA AREA = 1120
FREE SYMS = 0031
END- COMPILATION

B_TYPE MATSIZE.LIB
    %REPLACE
        MAXROW BY 26,
        MAXCOL BY 40;
```

Lista 9. Program INV na tej listi levo in zgoraj včitava s konzole koeficiente sistema hkratnih enačb in njihove desne strani. Vrstice 6 do 8, ki imajo za vrstičnimi številkami oznake +, so posledica INCLUDE stavka v izvirnem PLI programu; ta stavek včita na mesta teh vrstic zbirko MATSIZE.LIB . Vključitvena zbirka MATSIZE.LIB je prikazana na koncu te liste (glej zgoraj). Smisel vključitvene zbirke je v tem, da jo lahko vključujemo še v različne druge programe, kot bomo naredili kasneje, ko jo bomo vključili še v program INVERT (glej naslednjo listo). Polje koeficientov in rešitvenih vektorjev je določeno z MAT(26,40) v vrsticah 9 in 10. Vrsta PUT LIST stavkov izpiše sporočila na konzolo, kot je razvidno iz liste 11, z GET LIST stavkoma v vrsticah 36 in 43 pa preberemo s konzole koeficiente in rešitvene vektorje. V vrstici 46 pokličemo ENTRY proceduro INVERT (program za obrnitev matrike), ki je PLI program z liste 10, ki bo ločeno prevedena in kasneje povezana s proceduro INV na tej listi, ko bomo uporabili LINK ukaz. Preostali del programa natisne rešitve za spremenljivke A, B, C, ... sistema hkratnih enačb in inverzno matriko pri različnih rešitvenih vektorjih. Pri prekoračitvi obsega matrike 26 krat 40 se natisne sporočilo o prekoračitvi (vrstica 29), kot je nazorno razvidno iz rezultatne liste 11.

```
B_TYPE INVERT2.PRN

PL/I-80 V1.3  COMPILATION OF: INVERT2

D: DISK PRINT
L: LIST SOURCE PROGRAM

%INCLUDE 'MATSIZE.LIB';
   NO ERROR(S) IN PASS 1

   NO ERROR(S) IN PASS 2


PL/I-80 V1.3  COMPILATION OF: INVERT2

   1 A 0000 INVERT:
   2 A 0000      PROC (A,R,C);
   3+C 0000      %REPLACE
   4+C 000D          MAXROW BY 26,
   5+C 000D          MAXCOL BY 40;
   6 C 000D      DCL
   7 C 000D          (D, A(MAXROW,MAXCOL)) FLOAT (24),
   8 C 000D          (I,J,K,L,R,C) FIXED (6);
   9 C 000D      DO I = 1 TO R;
  10 C 0023      D = A(I,1);
  11 C 0042          DO J = 1 TO C - 1;
  12 C 0059          A(I,J) = A(I,J+1)/D;
  13 C 0082          END;
  14 C 00B2      A(I,C) = 1/D;
  15 C 00E4          DO K = 1 TO R;
  16 C 00FA          IF K ^= I THEN
  17 C 0104              DO;
  18 C 0104              D = A(K,1);
  19 C 0123                  DO L = 1 TO C - 1;
  20 C 013A                  A(K,L) = A(K,L+1) - A(I,L) * D;
  21 C 01B9                  END;
  22 C 01B9              A(K,C) = - A(I,C) * D;
  23 C 021C              END;
  24 C 021C          END;
  25 C 021C      END;
  26 A 021C  END INVERT;

CODE SIZE = 021C
DATA AREA = 0016
FREE SYMS = 02EA
END  COMPILATION
```

ime zbirke s programom INVERT. Nova zbirka INVERT. COM vsebuje povezavo in z njenim CP/M pozivom (preprosto INVERT) dobimo izvajanje INVERT programa v listi 11.

11. Procesiranje znakovnih nizov

Oglejmo si uporabo niznih funkcij, s katerimi obdelujemo nizne podatke. Napišimo PLI program z imenom OPTIMIST, ki daje na pesimistične izjave optimistične odgovore. Tu imamo program za omejeno področje naravnega jezika.

V listi 12 imamo program, kjer so v vrsticah 7 do 24 določeni podatki, ki se navajajo v programu. Preostali del programa čita stavke s konzole, ki se končujejo s piko ter jih izpiše v trdilni obliki. Primer izvajanja tega programa je pokazan v listi 13. Pri ustrezni obliki vhodnih stavkov se oblikujejo pravilni izhodni stavki.

Zaporedje nikalnih besed je definirano v vrsticah 8 do 11, temu sledi zaporedje nenikalnih (trdilnih) besed. Tako imamo razmerje: ne, -da; pozno - zgodaj; kakšen - - kolikšen; isla - isel; nikoli - vselej; nobeden - nekdo; nič - nekaj; ni - je; ne - ; nima - ima itd. Vključeni sta tudi abecedi velikih in malih črk (ter znakov) za prevajanje v delu programa, kjer se obdelujejo stavki. Neka-

Lista 10. Lista na levi strani kaže podprogram (neglavno PLI proceduro) za obrnitev matrike. Kot je bilo že opisano, so vrstice 3, 4 in 5 (znak +) posledica uporabe INCLUDE stavka v izvirnem PLI programu. Ta podprogram bo povezan z glavnim programom INV (glej prejšnjo listo) po uporabi LINK ukaza.

```
B_C_INVERT
SOLUTION OF SIMULTANEOUS EQUATIONS

TYPE ROWS, COLUMNS: 3,3

TYPE MATRIX OF COEFFICIENTS
ROW       1 :1,0,0
ROW       2 :0,1,0
ROW       3 :0,0,1

TYPE SOLUTION VECTORS


SOLUTIONS:
A =
B =
C =

INVERSE MATRIX IS
        1.00      0.00      0.00
        0.00      1.00      0.00
        0.00      0.00      1.00

TYPE ROWS, COLUMNS: 3,6

TYPE MATRIX OF COEFFICIENTS
ROW       1 :1,-1,1
ROW       2 :1,1,-1
ROW       3 :2,-1,0

TYPE SOLUTION VECTORS
VARIABLE A :2,0,8
VARIABLE B :3.5,1.5,-1.2
VARIABLE C :2,3,4


SOLUTIONS:
A =     1.00      2.50      2.50
B =     2.00      6.20      1.00
C =     3.00      7.20      0.50

INVERSE MATRIX IS
        0.50      0.50      0.00
        1.00      1.00     -1.00
        1.50      0.50     -1.00

TYPE ROWS, COLUMNS: 27,3

MATRIX IS TOO LARGE

TYPE ROWS, COLUMNS: 0

END OF EXECUTION
```

Lista 11. Lista zgoraj je rezultat izvajanja povezanih programov INV in INVERT z list 9 in 10. Ta povezava je bila izdana (generirana) kot nova CP/M zbirka z imenom INVERT, in sicer po prevodu in povezavi prvotnih zbirk INVERT1 in INVERT2 (listi 9 in 10).

```
PL/I-80 V1.3  COMPILATION OF: OPTIMIST

D: DISK PRINT
L: LIST SOURCE PROGRAM

   NO ERROR(S) IN PASS 1

   NO ERROR(S) IN PASS 2
```

```
PL/I-80 V1.3  COMPILATION OF: OPTIMIST

   1 A 0000 OPTIMIST:
   2 A 0006      PROC OPTIONS(MAIN);
   3 C 0006      %REPLACE
   4 C 0006          PRAVILNO     BY '1'B,
   5 C 0006          NEPRAVILNO   BY '0'B,
   6 C 0006          NBESED       BY 17;
   7 C 0006      DCL
   8 C 0006          NIKALNO (1:NBESED) CHAR(7) VAR STATIC INITIAL
   9 C 0006          (' NE,',' NE,',' NE',' NE',' NE',' NE',' NI ',
  10 C 0006           ' NI ',' NIS',' NIMA',' NIMA',' NIKOLI',
  11 C 0006           ' NIHCE',' NIC','POZNO','KAKSEN','ISLA'),
  12 C 0006          TRDILNO (1:NBESED) CHAR(8) VAR STATIC INITIAL
  13 C 0006          (' DA,',' DA,',' ',' ',' ',' ',' JE ',' JE ',
  14 C 0006           ' S',' IMA',' IMA',' VSELEJ',' NEKDO',
  15 C 0006           ' NEKAJ','ZGODAJ','KOLIKSEN','ISEL'),
  16 C 0006          VELIKE CHAR(32) STATIC INITIAL
  17 C 0006              ('ABCDEFGHIJKLMNOPQRSTUVWXYZ,;:-. '),
  18 C 0006          MALE   CHAR(32) STATIC INITIAL
  19 C 0006              ('ABCDEFGHIJKLMNOPQRSTUVWXYZ,;:-. ');
  20 C 0006      DCL
  21 C 0006          POSLANO CHAR(254) VAR,
  22 C 0006          BESEDA  CHAR(64)  VAR;
  23 C 0006      DCL
  24 C 0006          (I,J) FIXED;
  25 C 0006
  26 C 0006          DO WHILE(PRAVILNO);
  27 C 0006          PUT SKIP LIST('KAJ SE DOGAJA? ');
  28 C 0022          POSLANO = ' ';
  29 C 002F              DO WHILE
  30 C 008C                  ((SUBSTR(POSLANO,LENGTH(POSLANO)) ^= '.')&
  31 C 008C              (SUBSTR(POSLANO,LENGTH(POSLANO)) ^= '!')&
  32 C 008C              (SUBSTR(POSLANO,LENGTH(POSLANO)) ^= '?'));
  33 C 008C              GET LIST (BESEDA);
  34 C 00A6              POSLANO = POSLANO !! ' ' !! BESEDA;
  35 C 00C9              END;
  36 C 00C9          POSLANO = TRANSLATE(POSLANO,MALE,VELIKE);
  37 C 00EC          IF VERIFY(POSLANO,MALE) ^= 0 THEN
  38 C 0103              POSLANO = ' TO JE ZANIMIV PRIMER.';
  39 C 0110              DO I = 1 TO NBESED;
  40 C 0122              J = INDEX(POSLANO,NIKALNO(I));
  41 C 0137              IF J ^= 0 THEN
  42 C 0140                  POSLANO = SUBSTR(POSLANO,1,J-1) !!
  43 C 019D                     TRDILNO(I) !!
  44 C 019D                     SUBSTR(POSLANO,J+LENGTH(NIKALNO(I)));
  45 C 019D              END;
  46 C 019D          PUT LIST('KAJNE DA,' !! POSLANO);
  47 C 01BA          PUT SKIP;
  48 C 01CE          END;
  49 A 01CE      END OPTIMIST;

CODE SIZE = 01CE
DATA AREA = 02D7
FREE SYMS = 01B0
END  COMPILATION
```

Lista 12. Programska lista na levi strani in zgoraj opisuje program, ki daje na pesimistične odgovore uporabnika optimistične izjave v slovenskem jeziku. Ta program je tako poseben primer t.im. dialognega sistema ter nakazuje tako možnosti programiranja podobnih sistemov.

```
A_OPTIMIST

KAJ SE DOGAJA? ' NE,' NIHČE VAM NE BO ODGOVORIL.
KAJNE DA,   DA, NEKDO VAM  BO ODGOVORIL.

KAJ SE DOGAJA? KDOR POZNO 'VSTAJA,' NE DELA Z VESELJEM.
KAJNE DA,  KDOR ZGODAJ VSTAJA,  DELA Z VESELJEM.

KAJ SE DOGAJA? NIKOLI NISMO NIČ TAKEGA TRDILI: NIHČE NE ZNA.
KAJNE DA,  VSELEJ SMO NEKAJ TAKEGA TRDILI: NEKDO  ZNA.

KAJ SE DOGAJA? NEKAJ TI NE 'POVEM,' DA NE BOŠ MISLIL SLABO.
KAJNE DA,  KAJ TI  POVEM, DA  BOŠ MISLIL SLABO.

KAJ SE DOGAJA? KAKŠEN NESMISEL ČE NE BO SMISLA.
KAJNE DA,  KOLIKŠEN SMISEL ČE  BO SMISEL.

KAJ SE DOGAJA? TRISTO KOSMATIH!
KAJNE DA, TO JE ZANIMIV PRIMER.

KAJ SE DOGAJA? NESTROKOVNO NI ENAKO NEUSPEŠNEMU.
KAJNE DA,  STROKOVNO JE ENAKO USPEŠNEMU.

KAJ SE DOGAJA? ' NE,',,, NIMA,,, SMISLA,,,.
KAJNE DA,   DA,  DA,  , IMA IMA NIMA SMISEL SMISLA SMISLA .

KAJ SE DOGAJA? NIKOLI NISMO 'NEPRESPANI,' KER ZGODAJ VSTAJAMO.
KAJNE DA,  VSELEJ SMO PRESPANI, KER ZGODAJ VSTAJAMO.

KAJ SE DOGAJA? ^C
```

Lista 13. Ta lista kaže izvajanje programa, ki smo ga poklicali v izvrševanje z ukazom OPTIMIST s konzole. Program se javlja z vsakokratnim vprašanjem: Kaj se dogaja? Pesimističnemu odgovoru uporabniku sledi optimističen odgovor programa. V osmem primeru so prikazani tudi stranski učinki programa (uporaba več zaporednih vejic). Iz izpisanega teksta na listi je moč razpoznati tudi natančnejše delovanje programa.

tera razmerja se pojavljajo večkrat (npr. ne - prazno se pojavi štirikrat), zato da imamo v oblikovanem stavku večkratno možnost substitucije.

Program ne neha spraševati in DO zanka med vrsticami 26 in 48 se konča z znakom control-z ali s control-c na začetku vhodne vrstice. Izhodni stavek se oblikuje med vrsticami 28 do 32, ko DO zanka včitava besede ter jih stika na konec obstoječega teksta SUBSTR preizkus na začetku DO WHILE zanke preizkuša na piko, klicaj ali vprašaj, ki je na koncu stavka. Pri tem je maksimalna dolžina stavka enaka 254 znakov (dodatni znaki se ne upoštevajo).

Ko je bil stavek v celoti včitan, se vse velike črke prevedejo v male, tako da je mogoče razpoznati nikalne besede. Ta prevod se opravi z vgrajeno funkcijo TRANSLATE, kot kaže vrstica 36. V vrstici 37 se uporablja VERIFY funkcija, ki zagotavlja, da je stavek sestavljen samo iz črk, nekaterih znakov in pike. Če ta pogoj ni izpolnjen (npr. da imamo nekje "!"), odgovori program s stavkom

Kajneda, to je zanimiv primer.

Kadar vrne VERIFY funkcija vrednost nič, vsebuje stavek samo (prevedene) male črke, nekatere znake in piko. V tem primeru se izvrši DO skupina med vrsticama 39 in 45. Pri vsaki iteraciji se uporabi INDEX funkcija za iskanje naslednje nikalne besede, ki je določena z NIKALNO (I) (vrstica 40). Če je taka beseda najdena, se J postavi na položaj nikalne besede in se beseda v prireditvi vrstice 42 zamenja z ustrezno pozitivno (trdilno) besedo. V tej prireditvi je

SUBSTR (POSLANO, 1, J-1)

tisti del stavka, ki se je oblikoval pred nikalno besedo; nadalje je

TRDILNO (I)

nadomestna vrednost za nikalno besedo in

SUBSTR (POSLANO, J + LENGTH (NIKALNO (I)))

je del stavka, ki sledi nikalni besedi, ki je bila zamenjana. Stik teh treh segmentov je nova oblika stavka, kjer so bile vse (deklarirane) nikalne besede zamenjane z ustreznimi trdilnimi (nenikalnimi) besedami. Ker začenjajo vse nikalne besede z vodilnim presledkom, se lahko najdejo nikalni deli samo na začetkih besed. Tako se naprimer "nesposoben" zamenja s "sposoben" ali "ničvreden" z "nekajvreden" itn. Program OPTIMIST pošlje oblikovani stavek na konzolo ter čaka na naslednji stavek s konzole.

V programu OPTIMIST so mogoče tri izboljšave. Če prekoračimo pri tipkanju s konzole 254 znakov, se razpoznavanje vhodnega stavka ne bo končalo, ker program ne more najti pike. Zato je priporočljivo, da vgradimo v program preizkus, tako da ne prekoračimo maksimalne dolžine vhodnega stavka. Vključimo lahko tudi ON enoto, ki bo zaznala konec zbirke, tako da se bo program končal na primeren način. Seveda pa je mogoče predelati program OPTIMIST tudi tako, da bo zmogljivejši od obstoječega.

Iz programa OPTIMIST lahko brez večjih težav naredimo program PESIMIST, če v listi 12 medsebojno zamenjamo vrstice 9, 10, 11 z vrsticami 13, 14, 15.

12. Primeri z obdelavo zbirk

Obdelava zbirk sodi prav gotovo med najzanimivejše primere slehernega programirnega jezika. Uporabnik si skorajda vselej želi, da bi se tudi njegova konkretna obdelava podatkov shranila v obliki zbirke na disketi, čeprav dobiva rezultat na odgovore na konzoli ali tiskalniku. Ta ugotovitev velja še zlasti za kompleksne obdelave, ki jih ne želimo ponavljati in seveda za obdelave z velikim številom podatkov. Oglejmo si nekaj zanimivih primerov.

12.1. Kopiranje zbirk

Lista 14 prikazuje splošen PLI program za kopiranje zbirke v zbirko. Program določi dve zbirčni konstanti v vrstici 4, ki ju imenujemo VHOD in IZHOD. Zbirki se odpreta v vrstici 6 in 9. V vrstici 14 se začenja zanka, ki včitava podatke iz VHOD zbirke v VRSTICA vmesnik ter vpisuje od tod v IZHOD zbirko.

Oba OPEN stavka določata STREAM zbirko za ASCII podatke z notranjima vmesnikoma s po 8192 znaki. Prvi OPEN stavek vsebuje implicitno INPUT pridevek, drugi pa eksplicitno uporabi OUTPUT pridevek (sicer bi bila tudi ta zbirka tipa INPUT). TITLE del obeh OPEN stavkov po-

```
C_TYPE KOPIRAJ.PRN

PL/I-80 V1.3  COMPILATION OF: KOPIRAJ

D: DISK PRINT
L: LIST SOURCE PROGRAM

   NO ERROR(S) IN PASS 1

   NO ERROR(S) IN PASS 2

PL/I-80 V1.3  COMPILATION OF: KOPIRAJ

   1 A 0000 KOPIRAJ:
   2 A 0006      PROC OPTIONS(MAIN);
   3 C 0006      DCL
   4 C 0006          (VHOD,IZHOD) FILE;
   5 C 0006
   6 C 0006      OPEN FILE (VHOD) STREAM ENV(B(8192))
   7 C 0023          TITLE('$1.$1');
   8 C 0023
   9 C 0023      OPEN FILE (IZHOD) STREAM OUTPUT ENV(B(8192))
  10 C 0040          TITLE('$2.$2');
  11 C 0040      DCL
  12 C 0040          VRSTICA CHAR(254) VARYING;
  13 C 0040
  14 C 0040          DO WHILE('1'B);
  15 C 0040          READ  FILE (VHOD)  INTO (VRSTICA);
  16 C 0058          WRITE FILE (IZHOD) FROM (VRSTICA);
  17 C 0073          END;
  18 A 0073      END KOPIRAJ;

CODE SIZE = 0073
DATA AREA = 0109
FREE SYMS = 034B
END  COMPILATION
```

Lista 14. Lista na levi prikazuje program za kopiranje ene zbirke na drugo. V CP/M sistemu se kot zbirka smatrajo tudi podatki, ki se sprejemajo iz ali oddajajo v periferne naprave. Vhodna imenska zbirčna spremenljivka je $1.$1, izhodna pa $2.$2, ko se VHOD kopira na IZHOD.

veže notranje zbirčno ime z zunanjimi CP/M napravami in zbirkami. Prvo zbirčno ime postane tako ime zbirke, vtipkano s konzole, preden se program KOPIRAJ začne izvajati (oznaka $1.$1). Podobno se vtipka v ukazno vrstico tudi drugo ime (oznaka $2.$2), tako da imamo vobče direktivo:

KOPIRAJ IME1.TIP IME2 TIP

ko se zbirka IME1.TIP prepiše v zbirko IME2.TIP. Vhodna zbirka mora obstajati, izhodna zbirka pa se izbriše, če že obstaja in ponovno oblikuje.

Ta program kaže tudi posebno uporabo READ in WRITE stavka pri obdelavi STREAM zbirk: vrstica 15 bere STREAM zbirko v vmesnik VRSTICA, ki je dolžinsko spremenljivi znakovni niz. Vhodna vrstica do naslednjega vrstičnega pomika se včita v VRSTICA in dolžina spremenljivke VRSTICA se nastavi na včitano število podatkov, vključno z znakom za pomik vrstice (LF). Naslednji stavek opravi nasprotno dejavnost: WRITE stavek pošlje podatke iz VRSTICA v STREAM zbirko. Izhodna zbirka sprejme tako vse znake od začetka VRSTICA do trenutnega konca, ki je LENGTH (VRSTICA).

Program se konca, ko v vhodni zbirki včita STREAM konec, tj. znak CONTROL-z. V tej točki se vzpostavi END OF FILE stanje in izvajanje programa se ustavi. Vse zbirke se avtomatično zaprejo, spraznijo se notranji vmesniki in ohrani se novooblikovana izhodna zbirka.

Lista 15 kaže primer izvajanja kopirnega programa, ko imamo ukaz

KOPIRAJ KOPIRAJ.PLI $CON

Tu je KOPIRAJ.PLI vhodna zbirka, sistemska konzola je pa izhodna zbirka ($CON). Rezultat tega ukaza je listanje zbirke KOPIRAJ.PLI na terminalu. Npr. ukaz

KOPIRAJ D_X.DAT B_Y.NOV

bi prepisal zbirko X.DAT iz diskovnega pogona D v zbirko Y.NOV diskovnega pogona B.

```
D_KOPIRAJ KOPIRAJ.PLI $CON
KOPIRAJ:
    PROC OPTIONS(MAIN);
    DCL
        (VHOD,IZHOD) FILE;

    OPEN FILE (VHOD) STREAM ENV(B(8192))
        TITLE('$1.$1');

    OPEN FILE (IZHOD) STREAM OUTPUT ENV(B(8192))
        TITLE('$2.$2');
    DCL
        VRSTICA CHAR(254) VARYING;

        DO WHILE('1'B);
        READ  FILE (VHOD)  INTO (VRSTICA);
        WRITE FILE (IZHOD) FROM (VRSTICA);
        END;
    END KOPIRAJ;

END OF FILE (3), FILE: VHOD=KOPIRAJ.PLI
TRACEBACK: 044B 03AF 0155
END OF EXECUTION
```

Lista 15. Gornja lista kaže primer uporabe programa KOPIRAJ iz liste 14, ko se vhodna zbirka, t.j. izvirni program KOPIRAJ.PLI, kopira na konzolo ($CON). Zbirčna imenska parametra se pojavita v ukazni vrstici.

```
A_OBLIKUJ
PROGRAM ZA OBLIKOVANJE IMEN IN
   NASLOVOV... IME ZBIRKE : NASLOVI.NAS


IME (12): 'DR. ANTON P.
PRIIMEK (22): ZELEZNIKAR
ULICA S STEVILKO (APOSTOF)(30): 'PARMOVA C. 41/II
POST.STEVILKA (10): 61000
KRAJ (25): LJUBLJANA
DRZAVA (25): SFRJ
TELEFON (15): '(061) 312-988
TELEKS (15): '31366 YU DELTA


IME (12): 'PROF. JOHN F.
PRIIMEK (22): ARROWSMITH
ULICA S STEVILKO (APOSTOF)(30): 'THIRD AVE. 5544
POST.STEVILKA (10): 01234
KRAJ (25): 'NEW YORK
DRZAVA (25): 'NEW YORK, U.S.A.
TELEFON (15): XYZ-1234567890
TELEKS (15): 98765432-W


IME (12): ADAM
PRIIMEK (22): JABOLKO
ULICA S STEVILKO (APOSTOF)(30): 'V DOLINI 25
POST.STEVILKA (10): 69999
KRAJ (25): 'CRNI VRH NAD LOKO
DRZAVA (25): JUGOSLAVIJA
TELEFON (15): 061-22-33-44
TELEKS (15): 10001-YU-ANTENA


IME (12): KONEC

END OF EXECUTION
```

Lista 17. Program OBLIKUJ z liste 16 (na naslednji strani) se izvaja v sodelovanju s konzolo, ko se vstavi (določi) ime zbirke, nato pa se vstavljajo še podatki, dokler se ne pojavi ime KONEC.

12.2. Obdelava imenskih in naslovnih zbirk

V tem podpoglavju si oglejmo dva uporabna programa, ki sicer za oblikovanje (OBLIKUJ) in razpoznavanje (RAZPOZNA) zbirke imen in naslovov. OBLIKUJ program naj oblikuje STREAM zbirko posameznih imen in naslovov, dostop do teh podatkov pa imejmo prek programa RAZPOZNA.

OBLIKUJ program v listi 16 vsebuje podatkovno strukturo ZAPIS, ki s svojim formatom določa ime, priimek, ulico, poštno številko, kraj, državo, telefon in teleks. Prek konzole se vpiše vsak od navedenih podatkov v zbirko, dokler uporabnik ne vtipka imena KONEC. Seveda morata imeti oba programa (OBLIKUJ in RAZPOZNA) enako ZAPIS strukturo. Zato bi lahko to strukturo definirali s posebno zbirko (na disketi) npr. z imenom ZAPIS.DCL, v oba programa pa bi jo vključili na ustreznih mestih s stavkom

%INCLUDE ' ZAPIS.DCL';

V programu liste 16 se vstavi ime zbirke prek konzole s stavkom v vrstici 26, pri odprtju zbirke (vrstici

```
A_TYPE OBLIKUJ.PRN

PL/I-80 V1.3  COMPILATION OF: OBI

D: DISK PRINT
L: LIST SOURCE PROGRAM

   NO ERROR(S) IN PASS 1

   NO ERROR(S) IN PASS 2

PL/I-80 V1.3  COMPILATION OF: OBI

   1    0000 /*************************************************
   2    0000 *      PRIMER #9,  A.P.ZELEZNIKAR                *
   3    0000 *      DATUM: 27.4.1981                          *
   4    0000 *      OBLIKOVANJE ZBIRKE IMEN IN NASLOVOV       *
   5    0000 *************************************************/
   6    0000
   7 A  0000 OBLIKUJ: PROCEDURE OPTIONS (MAIN);
   8 C  0006   DCL 1 ZAPIS,
   9 C  0006         2 IME CHAR(12) VAR,
  10 C  0006         2 PRIIMEK CHAR(22) VAR,
  11 C  0006         2 ULICA CHAR(30) VAR,
  12 C  0006         2 POSTNA_STEVILKA CHAR(10) VAR,
  13 C  0006         2 KRAJ CHAR(25) VAR,
  14 C  0006         2 DRZAVA CHAR(25) VAR,
  15 C  0006         2 TELEFON CHAR(15) VAR,
  16 C  0006         2 TELEKS CHAR(15) VAR;
  17 C  0006   %REPLACE NEPRAVILNO BY '0'B,
  18 C  0006            PRAVILNO BY '1'B;
  19 C  0006    DCL IZHOD FILE;
  20 C  0006    DCL IME_ZBIRKE CHAR(14) VARYING;
  21 C  0006    DCL KONEC_ZBIRKE BIT(1) STATIC INITIAL
  22 C  0006                                 (NEPRAVILNO);
  23 C  0006
  24 C  0006    PUT LIST('PROGRAM ZA OBLIKOVANJE IMEN IN');
  25 C  001D    PUT SKIP LIST('   NASLOVOV... IME ZBIRKE : ');
  26 C  0039    GET LIST(IME_ZBIRKE);
  27 C  0053    OPEN FILE(IZHOD) STREAM OUTPUT TITLE
  28 C  006D                              (IME_ZBIRKE);
  29 C  006D       DO WHILE (^KONEC_ZBIRKE);
  30 C  0074       PUT SKIP(3) LIST('IME (12): ');
  31 C  0090       GET LIST(IME);
  32 C  00AA       KONEC_ZBIRKE = (IME = 'KONEC');
  33 C  00BC       IF ^KONEC_ZBIRKE  THEN
  34 C  00C3         DO;
  35 C  00C3         /* IZPISI NAVODILA ZA VPIS S KONZOLE */
  36 C  00C3         PUT LIST('PRIIMEK (22): ');
  37 C  00DA         GET LIST(PRIIMEK);
  38 C  00F4         PUT LIST('ULICA S STEVILKO (APOSTOF)(30): ');
  39 C  010B         GET LIST(ULICA);
  40 C  0125         PUT LIST('POST.STEVILKA (10): ');
  41 C  013C         GET LIST(POSTNA_STEVILKA);
  42 C  0156         PUT LIST('KRAJ (25): ');
  43 C  016D         GET LIST(KRAJ);
  44 C  0187         PUT LIST('DRZAVA (25): ');
  45 C  019E         GET LIST(DRZAVA);
  46 C  01B8         PUT LIST('TELEFON (15): ');
  47 C  01CF         GET LIST(TELEFON);
  48 C  01E9         PUT LIST('TELEKS (15): ');
  49 C  0200         GET LIST(TELEKS);
  50 C  021A
  51 C  021A     /* ZAPISI PODATKE IZ POMN. V ZBIRKO IZHOD */
  52 C  021A         PUT FILE(IZHOD)
  53 C  0271          LIST(IME,PRIIMEK,ULICA,POSTNA_STEVILKA,
  54 C  0271                KRAJ,DRZAVA,TELEFON,TELEKS);
  55 C  0271         PUT FILE(IZHOD) SKIP;
  56 C  0288         END;
  57 C  0288       END;
  58 C  0288    PUT FILE(IZHOD) SKIP LIST('KONEC');
  59 C  02A7    PUT FILE(IZHOD) SKIP;
  60 A  02BB    END OBLIKUJ;

CODE SIZE = 02BE
DATA AREA = 0170
FREE SYMS = 0080
END  COMPILATION
```

Lista 16. Program na levi strani in zgoraj oblikuje zbirko imen in naslovov, kot je razvidno iz list 17, 18 in 20. Ta zbirka se oblikuje v sodelovanju s konzolo, ko se izbere ime zbirke, potem pa se vstavljajo podatki. Ti podatki so določeni s podatkovno strukturo ZAPIS (vrstica 8), ki jo sestavljajo člani IME, PRIIMEK, ULICA, POŠTNA_ŠTEVILKA, KRAJ, DRŽAVA, TELEFON in TELEKS (vrstice 9 do 16). Ti podatki se zlagajo v imenovano zbirko (IZHOD) v t.im. STREAM formatu (vrstica 27, kjer se zbirka odpre in vrstici 53 in 54, kjer se podatki v njo naložijo). Pred avtomatičnim zaprtjem zbirke se naloži v njo še ime KONEC in konec vrstice, nakar se zbirka pred končanjem programa zapre.

```
A_TYPE NASLOVI.NAS
'DR. ANTON P.' 'ZELEZNIKAR' 'PARMOVA C. 41/II' '61000' 'LJUBLJANA' 'SFRJ
'(061) 312-988' '31366 YU DELTA'
'PROF. JOHN F' 'ARROWSMITH' 'THIRD AVE. 5544' '01234' 'NEW YORK'
'NEW YORK, U.S.A.' 'XYZ-1234567890' '98765432-W'
'ADAM' 'JABOLKO' 'V DOLINI 25' '69999' 'CENI VRH NAD LOKO' 'JUGOSLAVIJA'
'061-22-33-44' '10001-YU-ANTENA'

'KONEC'
```

Lista 18. Oblikovana zbirka NASLOVI.NAS je bila dobljena z izvajanjem programa OBLIKUJ z liste 17. Kot je razvidno, je zbirka tipa STREAM in vsak element ZAPISa je samostojen znakovni niz. Vsak zapis je sestavljen natanko iz osmih, po dolžini spremenljivih nizov. Ta podatkovni format bo lahko upoštevan tudi v programu za razpoznavanje v listi 19. Zbirka NASLOV.NAS je bila izpisana s TYPE ukazom in njen prikaz je natančno njena slika na disketi.

```
PL/I-80 V1.3  COMPILATION OF: RAZPOZNA

D: DISK PRINT
L: LIST SOURCE PROGRAM

   NO ERROR(S) IN PASS 1

   NO ERROR(S) IN PASS 2


PL/I-80 V1.3  COMPILATION OF: RAZPOZNA

   1    0000    /****************************************
   2    0000    *  PRIMER #6, A.P.ZELEZNIKAR            *
   3    0000    *  DATUM: 31.3.1981                     *
   4    0000    *  RAZPOZNAVANJE IN TISKANJE IZ ZBIRKE  *
   5    0000    *    IMEN IN NASLOVOV                   *
   6    0000    ****************************************/
   7    0000
   8 A  0000 RAZPOZNA: PROCEDURE OPTIONS (MAIN);
   9 A  0006
  10 C  0006   DCL 1 ZAPIS,
  11 C  0006         2 IME CHAR (12) VAR,
  12 C  0006         2 PRIIMEK CHAR (22) VAR,
  13 C  0006         2 ULICA CHAR (30) VAR,
  14 C  0006         2 POSTNA_STEVILKA CHAR (10) VAR,
  15 C  0006         2 KRAJ CHAR (25) VAR,
  16 C  0006         2 DRZAVA CHAR (25) VAR,
  17 C  0006         2 TELEFON CHAR (15) VAR,
  18 C  0006         2 TELEKS CHAR (15) VAR;
  19 C  0006
  20 C  0006   %REPLACE
  21 C  0006         PRAVILNO  BY  '1'B,
  22 C  0006         NEPRAVILNO  BY  '0'B;
  23 C  0006
  24 C  0006   DCL (SYSPRINT, VHOD) FILE;
  25 C  0006   DCL IME_ZBIRKE CHAR (14) VAR,
  26 C  0006       (SPODNJI, ZGORNJI) CHAR (30) VAR,
  27 C  0006       KONEC_ZBIRKE BIT (1);
  28 C  0006
  29 C  0006   OPEN FILE (SYSPRINT) PRINT TITLE ('$CON');
  30 C  0022   PUT LIST ('RAZPOZNAVANJE IMEN IN NASLOVOV');
  31 C  0039   PUT SKIP LIST ('   IME ZBIRKE: ');
  32 C  0055   GET LIST (IME_ZBIRKE);
  33 C  006F   DO WHILE (PRAVILNO);
  34 C  006F   SPODNJI = 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
  35 C  007B   ZGORNJI = 'ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ';
  36 C  0087   PUT SKIP (2) LIST (
  37 C  00A3     'VPISI SPODNJO IN ZGORNJO MEJO: ');
  38 C  00A3   GET LIST (SPODNJI, ZGORNJI);
  39 C  00CB   IF SPODNJI = 'KONEC' THEN
  40 C  00D9     STOP;
  41 C  00DC
  42 C  00DC   OPEN FILE (VHOD) STREAM
  43 C  00F7       INPUT ENVIRONMENT (B(1024))
  44 C  00F7       TITLE (IME_ZBIRKE);
  45 C  00F7   KONEC_ZBIRKE = NEPRAVILNO;
  46 C  00FC     DO WHILE (^KONEC_ZBIRKE);
  47 C  0103     GET FILE (VHOD) LIST (IME);
  48 C  0120     KONEC_ZBIRKE = (IME = 'KONEC');
  49 C  0132     IF ^KONEC_ZBIRKE THEN
  50 C  0139       DO;
  51 C  0139       GET FILE (VHOD) LIST
  52 C  01AA         (PRIIMEK, ULICA, POSTNA_STEVILKA,
  53 C  01AA           KRAJ, DRZAVA, TELEFON, TELEKS);
  54 C  01AA       IF PRIIMEK >= SPODNJI &
  55 C  01C7          PRIIMEK <= ZGORNJI THEN
  56 C  01C7          DO;
  57 C  01C7          PUT SKIP (3)
  58 C  01EA            LIST (IME,PRIIMEK);
  59 C  01EA           PUT SKIP LIST (ULICA);
  60 C  0204           PUT SKIP LIST (POSTNA_STEVILKA,' ',KRAJ);
  61 C  0232           PUT SKIP LIST (DRZAVA);
  62 C  024C           PUT SKIP LIST ('TELEFON:', TELEFON);
  63 C  0271           PUT SKIP LIST ('TELEX:', TELEKS);
  64 C  0299          END;
  65 C  0299       END;
  66 C  0299     END;
  67 C  0299   CLOSE FILE (VHOD);
  68 C  02A2   END;
  69 A  02A2 END RAZPOZNA;

CODE SIZE = 02A2
DATA AREA = 0190
FREE SYMS = 006D
END  COMPILATION
```

Lista 19. Program RAZPOZNA na levi, rabi za razpoznavanje (izpisovanje) želienih podatkov iz poljubne zbirke, ki je bila oblikovana s programom OBLIKUJ. Ta program je natančneje opisan v tekstu.

```
A_RAZPOZNA
RAZPOZNAVANJE IMEN IN NASLOVOV
   IME ZBIRKE: NASLOVI.NAS

VPISI SPODNJO IN ZGORNJO MEJO: ,,


DR. ANTON P. ZELEZNIKAR
PARMOVA C. 41/II
61000    LJUBLJANA
SFRJ
TELEFON: (061) 312-988
TELEX: 31366 YU DELTA


PROF. JOHN F ARROWSMITH
THIRD AVE. 5544
01234    NEW YORK
NEW YORK, U.S.A.
TELEFON: XYZ-1234567890
TELEX: 98765432-W


ADAM JABOLKO
V DOLINI 25
69999    CRNI VRH NAD LOKO
JUGOSLAVIJA
TELEFON: 061-22-33-44
TELEX: 10001-YU-ANTENA

VPISI SPODNJO IN ZGORNJO MEJO: A,I


PROF. JOHN F ARROWSMITH
THIRD AVE. 5544
01234    NEW YORK
NEW YORK, U.S.A.
TELEFON: XYZ-1234567890
TELEX: 98765432-W

VPISI SPODNJO IN ZGORNJO MEJO: KONEC
,,

END OF EXECUTION
```

Lista 20. Zgoraj vidimo izvajanje programa RAZPOZNA. Ko vpišemo s konzole ime zbirke ter spodnjo in zgornjo mejo (glede na priimek), se podatki izpišejo. Pri dveh vejicah na meji se izpiše celotna zbirka, pri mejah A, I pa samo priimki med A in I. Program končamo, ko vtipkamo na meji KONEC in dve vejici.

27 in 28) pa postane to ime tudi ime odprte zbirke z uporabo TITLE pridevka. Tu ni uporabljen PRINT pridevek, kar omogoča kasnejšo uporabo zbirke za vhod podatkov, ko se uporabi GET LIST stavek.

Interakcija programa OBLIKUJ s konzolo je prikazana z listo 17. V našem primeru je izhodna zbirka poimenovana z NASLOVI.NAS (tretja vrstica liste 17). Pri tem moramo upoštevati, da so LIST vhodi (vrstice 31, 37, 39, ..., 49 liste 16) vobče omejeni s presledkom, ali vejico, ali s pomikom valja, če ti ločilni znaki niso v okviru niznih narekovajev. Tako so npr. 'PARMOVA C.41/II, 'PROF. JOHN F. in 'ČRNI VRH NAD LOKO nizi, ki se smatrajo kot en sam podatek, prirejen določeni vhodni spremenljivki. V oklepajih liste 17 so vpisana največja dopustna števila znakov v posameznih nizih, tako da lahko uporabnik te omejitve upošteva. Če bi npr. uporabnik pri spremenljivki KRAJ vpisal

ČRNI VRH NAD LOKO

potem bi postal KRAJ ČRNI, DRŽAVA VRH, TELEFON NAD in TELEKS LOKO. Pri pomiku valja se avtomatično vstavi drugi nizni narekovaj.

S CP/M TYPE ukazom lahko pogledamo, kako se podatki z liste 17 zapišejo v zbirko NASLOVI.NAS; ta primer je prikazan z listo 18. STREAM zaporedje je sestavljeno iz nizov v narekovajih, vsak zapis je sestavljen iz 8 polj (nizov) spremenljive dolžine; ta polja imajo natanko določen pomen, kot je določeno s podatkovno strukturo ZAPIS liste 16 (vrstice 8 do 16).

Program RAZPOZNA z liste 19 prečita s programom OBLIKUJ oblikovano zbirko ter izlista ali prikaže na zaslonu podatke skladno z uporabnikovo zahtevo. Podatkovna struktura ZAPIS programa RAZPOZNA je enaka oni za program OBLIKUJ, tako da imamo natančno ujemanje podatkov v enem in drugem programu.

Razpoznavni program deluje takole: glavna zanka med vrsticami 33 do 68 liste 19 včita dve nizni vrednosti (vrstica 38) skladno z najnižjim in najvišjim željenim priimkom (oziroma začetnim znakom ali začetno skupino znakov). Skladno s to zahtevo se bodo izpisali posamezni zapisi. Vgnezdena zanka med vrsticami 46 do 66 včita celotno vhodno zbirko ter izpiše le zapise s priimki med spodnjo in zgornjo mejo.

Podobno kot oblikovalni program včita razpoznavni program ime izvirne zbirke s konzole ter odpre in zapre to zbirko vsakokrat, ko se s konzole pojavi razpoznavna zahteva. OPEN stavek v vrstici 42 pripravi vhodno zbirko z notranjim vmesnikom dolžine 1024 zlogov. Ko je bila zbirka obdelana, se izvrši CLOSE stavek v vrstici 67 in notranji vmesniki se sprostijo. Vhodna zbirka se oblikuje vsakokrat na začetku razpoznavne zahteve.

Izvajanje programa RAZPOZNA kaže lista 20, kjer je NASLOVI.NAS ime prej oblikovane zbirke. Pri spodnji in zgornji meji prazno (znak ,, na listi 20) se izpišejo vsi zapisi, pri intervalu

A, I

pa samo tisti od A do I (v našem primeru Arrowsmith). Če namreč ne postavimo spodnje in gornje meje, velja inicializacija v vrsticah 34 in 35 liste 19, kjer dobi spremenljivka ZGORNJI vrednost, ki je enaka nizu tridesetih malih črk "z". S tem je omejeno celotno abecedno področje (priimkov).

Omenimo še, da je bila SYSPRINT zbirka odprta eksplicitno s PRINT pridevkom v vrstici 29, da smo tako nazorno pokazali obliko rezultirajočega izhoda. Ta stavek je namreč odvečen, ker ima PUT stavek v vrstici 30 tudi ta učinek.

13. Uporaba PICTURE formata

V tem poglavju si na kratko oglejmo t.im. PICTURE format izhodnih podatkov, ki je implementiran v jeziku PL/I-80. Ta format je usklajen z ANSI PL/I standardom (Subset G).

13.1. Sintaksa PICTURE formata

PICTURE podatkovni format se uporablja za urejevanje numeričnih podatkov s fiksno decimalno vejico. Vrednost takega urejevanja je znakovni niz, katerega oblika je določena z numerično vrednostjo in s PICTURE specifikacijo. Sintaksa tega formata je

P ⟨picture specifikacija⟩

kjer je picture specifikacija nizna konstanta, ki določa PICTURE specifikacijo. Tak format se lahko uporablja v PUT EDIT stavku podobno kot drugi formati. Znakovna nizna konstanta za opis PICTURE specifikacije je sestavljena iz enega ali več naslednjih znakov

$ + - S	statični ali pomični znaki
* Z	pogojna številčna znaka
9	številčni znak
V	znak položaja decimalne vejice
/ , . : B	vstavitveni znaki
CR DB	kreditni in dolžniški znak

ter mora izpolnjevati določena sintaksna pravila. Vstavitveni znaki se lahko pojavijo kjerkoli v veljavni ⟨picture specifikacija⟩ , toda ne smejo ločevati znakov PICTURE znakovnih parov CR in DB. Če odstranimo vse vstavitvene znake iz specifikacije, mora biti preostali niz sprejemljiv za (nedeterministični) končni razpoznavnik na sliki 3. Neoznačene poti pomenijo prehode, ko nobenega znaka v specifikacijskem nizu ne upoštevamo. Veljavne PICTURE specifikacije so npr.:

```
'B B $***,***V.99BB'
'S - - - - , 999V.99BCR'
'99 : 99 : 99'
'**/**/**'
' : B B B S S S S S, S S S.V S S B B B :'
```

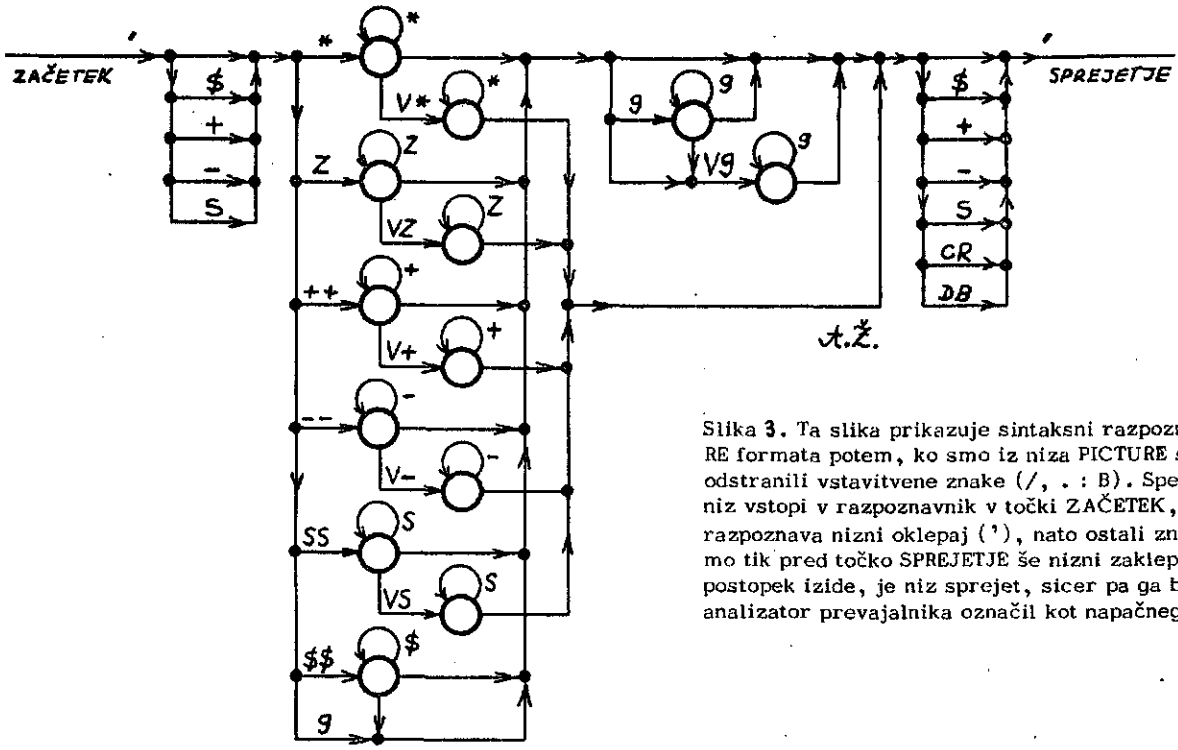
13.2. Semantika PICTURE formata

PICTURE specifikacija uredi numerično vrednost v znakovno nizno vrednost z uporabo posameznih tipov PICTURE znakov v specifikaciji. Znaki '$', '+', '-' in 'S' so statični ali pomični znaki. Tak znak je statičen, če se pojavi samo enkrat v PICTURE specifikaciji, sicer je pomičen. Pri pomičnem znaku ustrezajo vse njegove pojavitve z izjemo ene pogojnim številčnim položajem. PICTURE znaki določajo izhodni znak skupaj s predznakom numerične vrednosti in ta znak je dan s tabelo 1 in zaseda en sam položaj v izhodnem podatku.

	statični / nosilni znaki			
znak	S	+	-	$
poz.	+	+	' '	$
neg.	-	' '	-	$

Tabela 1

Če je PICTURE znak statičen, se bo izhodni znak pojavil na ustreznem položaju (mestu) izhoda. Če je PICTURE podatek pomičen, se bo izhodni znak pojavil natanko eno mesto pred neničto številko, nad katero se PICTURE



Slika 3. Ta slika prikazuje sintaksni razpoznavnik PICTURE formata potem, ko smo iz niza PICTURE specifikaciji odstranili vstavitvene znake (/, . : B). Specifikacijski niz vstopi v razpoznavnik v točki ZAČETEK, ko se najprej razpoznava nizni oklepaj ('), nato ostali znaki, ko imamo tik pred točko SPREJETJE še nizni zaklepaj ('). Če se postopek izide, je niz sprejet, sicer pa ga bo sintaksni analizator prevajalnika označil kot napačnega.

podatek pomakne, ali pa se bo pojavil nad zadnjim mestom, nad katerim se pomika. Vse druge pojavitve pomičnega znaka bodo nadomeščane s presledki skladno z opustitvijo ničtih številk v numerični vrednosti.

Znaka '*' in 'Z' imenujemo pogojni številki PICTURE znakov ali znaki za opustitev ničle. Vsak tak podatek je v PICTURE specifikaciji povezan s številko numerične vrednosti. V izhodu dobimo za vsako ustrezno številko nič izhodni znak '*' ali ' '. Če je ustrezna številka neničelna, se na izhodu pojavi številčni znak.

PICTURE znaki 'B', '/', '.', ':' in ',' se imenujejo vstavitveni (znak ':' ni vstavitven po ANSI standardu, vendar je bil dodan v PL/I-80 za prikazovanje numeričnih podatkov, ki predstavljajo čas). Vstavitveni karakterji se preslikajo sami vase na ustrezne položaje izhodnega podatka (B se preslika v presledek), če se pri tem kakšen vstavitveni znak ne pojavi v polju pomičnega znaka ali znaka za opustitev ničle. Kadar se vstavitveni znak pojavi v polju pomičnega znaka ali znaka za opustitev ničle, ki bi povzročil opustitev numerične številke, se opusti tudi vstavitveni znak skladno s prejšnjimi pravili.

V nekaterih implementacijah je 'B' brezpogojni vstavitveni znak in povzroči vselej presledek na ustreznem položaju izhodnega podatka. Po ANSI standardu se lahko takšen presledek prepiše (nadpiše) s pomičnim znakom ali z znakom '*' za opustitev ničle.

PICTURE znak '9' v PICTURE specifikaciji določa, da se bo ustrezna številka numerične vrednosti pojavila na ustreznem mestu izhoda. Tako je '9' brezpogojni številčni položaj.

Korespondenca (relacija) med številkami numerične vrednosti v PICTURE specifikaciji se določa s PICTURE znakom 'V'. Ta znak rabi za določitev mesta, kjer se končuje celi del in se začenjajo številke ulomljenega dela in tako določa razvrstitev znakov v numerični vrednosti glede na PICTURE specifikacijo. Če se ta znak ne pojavi, se predpostavlja, da se vsi številčni položaji PICTURE specifikacije nanašajo na številke celega dela (ne ulomljenega) in se zaradi tega v numerični vrednosti ne bodo pojavile na izhodu številke ulomljenega dela. Pri tem PICTURE znak 'V' kot edini ne vpliva na kakšen znakovni položaj rezultata. Tako je dolžina rezultata enaka dolžini PICTURE specifikacije, v kateri se 'V' ne pojavi, toda je za en znak krajša, če se 'V' pojavi. Očitno ima 'V' tudi učinek opustitve znakov. Ulomkovne številke, ki ustrezajo položajem za znakom 'V' se tako ne opustijo, če se ne opustijo vse ulomkovne številke. Splošno velja, da se za znakom 'V' uvede ukinitev opustitve (torej se številke pišejo), če je poprej opustitev veljala. Vstavitveni znak za znakom 'V', kot je npr. decimalna pika, se ne opusti, če se ne opustijo vsi znaki za 'V'.

Znaka 'CR' in 'DB' pomenita 'kredit' in 'dolg' in sta v bistvu predznaka. Če se eden od njiju pojavi v PICTURE specifikaciji in je predznak numerične vrednosti negativen, potem se bo specificirani par pojavil v rezultatu. Če pa je numerična vrednost pozitivna, se bosta na ustreznih mestih za ta znakovni par pojavila dva presledka (to se vidi v kasnejšem primeru).

Dodatno k navedenim pravilom, obstajajo tudi izjeme. Če je numerična vrednost enaka nič in če PICTURE specifikacija ne vsebuje znaka '9', se bodo v izhodnem podatku pojavili sami znaki '*', če se seveda '*' pojavlja kot PICTURE znak; sicer se bodo pojavili v izhodnem podatku sami presledki. To pravilo je prednostno glede na prejšnja pravila. Kadar je predznak numerične vrednosti negativen in če ni nobeden od PICTURE znakov v specifikaciji enak znakom S, +, -, CR ali DB, se bo pojavilo sporočilo ERROR (1) ali sporočilo o konverzijski napaki.

Vsaka PICTURE specifikacija povzroči natančnost in obseg numerične vrednosti v rezultatu skladno z naslednjimi pravili: Vstavitveni znaki in znakovni pari za CR in DB ne vplivajo na natančnost in obseg. Razen tega bo natančnost rezultata za eno manjša kot je število statičnih/pomičnih znakov ali kot je število znakov za opustitev ničle plus število znakov '9'. Obseg rezultata bo nič, če se 'V' ne pojavi. Če pa se 'V' pojavi, bo obseg rezultata enak številu pomičnih znakov ali številu znakov za opustitev ničle ali številu znakov '9' za znakom 'V'.

Lista 21. Program (oziroma zbirka) PIC1.PLI na desni določa osem vhodnih vrednosti A(I) (I = 1, ... ,8), ki jih bomo med izvajanjem programa vtipkali s konzole, tri formatne vrednosti B(J) (J = 1,2,3) in nizno vrednost C (Din). Vsaka vtipkana vrednost se bo izpisala v treh formatih, kot določajo trije PUT EDIT stavki.

```
C_TYPE PIC1.PLI
PIC1:
  PROCEDURE OPTIONS(MAIN);
  DCL
    A(1:8) FIXED DEC(10,2) STATIC,
    B(1:3) CHAR(20) VAR STATIC INIT
      ('BBS**,***.***V,99BB',
       '$**.***,***V**CR','/+++,+++.+++,V++/'),
    C CHAR(3) STATIC INIT ('DIN'),
    I FIXED;

  PUT SKIP LIST ('VPISI 8 VREDNOSTI: ');
  PUT SKIP;
  DO I=1 TO 8;
    PUT LIST ('VREDNOST ',I,'(FORMAT 10,2): ');
    GET LIST (A(I));
  END;

  PUT SKIP LIST ('PRIMERI PICTURE PODATKOV:');
  DO I=1 TO 8;
    PUT SKIP EDIT (A(I),B(1),C,A(I))
      (F(10,2),COL(15),A(20),A(3),P'BBS**,***.***V,99BB');
    PUT SKIP EDIT (A(I),B(2),A(I))
      (F(11,3),COL(15),A(20),COL(41),P'$**.***,***V**CR');
    PUT SKIP EDIT (A(I),B(3),C,A(I))
      (F(10,2),COL(15),A(20),A(3),P'/++,+++.+++,V++/');
  END;
END PIC1;
```

```
D_PIC1

VPISI 8 VREDNOSTI:
VREDNOST          1 (FORMAT 10,2): 0.00
VREDNOST          2 (FORMAT 10,2): -0.12
VREDNOST          3 (FORMAT 10,2): 1.23
VREDNOST          4 (FORMAT 10,2): -12.34
VREDNOST          5 (FORMAT 10,2): 123.45
VREDNOST          6 (FORMAT 10,2): -1234.56
VREDNOST          7 (FORMAT 10,2): 12345.67
VREDNOST          8 (FORMAT 10,2): -123456.78

PRIMERI PICTURE PODATKOV:
      0.00    BBS**,***.***V,99BB DIN  +**********,00
      0.000   $**.***,***V**CR         ***************
      0.00    /+++,+++.+++,V++/   DIN
     -0.12    BBS**,***.***V,99BB DIN  -**********,12
     -0.120   $**.***,***V**CR          $**********12CR
     -0.12    /+++,+++.+++,V++/   DIN/        12/
      1.23    BBS**,***.***V,99BB DIN  +*********1,23
      1.230   $**.***,***V**CR          $*********123
      1.23    /+++,+++.+++,V++/   DIN/       +1,23/
    -12.34    BBS**,***.***V,99BB DIN  -********12,34
    -12.340   $**.***,***V**CR          $********1234CR
    -12.34    /+++,+++.+++,V++/   DIN/      12,34/
    123.45    BBS**,***.***V,99BB DIN  +*******123,45
    123.450   $**.***,***V**CR          $*******12345
    123.45    /+++,+++.+++,V++/   DIN/     +123,45/
  -1234.56    BBS**,***.***V,99BB DIN  -*****1.234,56
  -1234.560   $**.***,***V**CR          $*****1,23456CR
  -1234.56    /+++,+++.+++,V++/   DIN/    1.234,56/
  12345.67    BBS**,***.***V,99BB DIN  +****12.345,67
  12345.670   $**.***,***V**CR          $****12,34567
  12345.67    /+++,+++.+++,V++/   DIN/   +12.345,67/
-123456.78    BBS**,***.***V,99BB DIN  -***123.456,78
-123456.780   $**.***,***V**CR          $***123,45678CR
-123456.78    /+++,+++.+++,V++/   DIN/   123.456,78/
END OF EXECUTION
```

Lista 22. Lista na levi kaže izvajanje programa PIC1. Najprej se pojavlja osem s konzole vpisanih vrednosti, ki smo jim izmenično spreminjali predznak. Za vhodno vrednost -123456.78 dobimo tri formatirane rezultate, in sicer -***123.456,78, $***123,45678CR in / 123.456,78/. V vsaki vrstici imamo vrednost s konzole (format F(10,2) ali F(11,3)) ter formatno in formatizirano vrednost.

13.3. Primer uporabe PICTURE formata

PICTURE format se uporablja za prikazovanje poslovnih podatkov. Možnosti njegove uporabe so zelo raznovrstne, kot je razvidno iz sintaksnega diagrama na sliki 3.

Lista 21 kaže program za izpis treh različnih PICTURE formatov za vsakokratnih osem vrednosti, ki jih vpišemo s konzole. Lista 22 kaže izvajanje programa PIC1. Podrobnejša pojasnila so dana v opisih list 21 in 22.

Dodatna literatura

(20) A.Schulz, Höhere PL/1-Programmierung, Zal. Walter de Gruyter, Berlin, New York, 1976 (214 strani, mehkovezana, visok stavek).

Knjiga opisuje višje, vsekakor pa posebne, izbrane primere programiranja v jeziku PL/1. Najprej uvaja formalna jezikovno sintakso, in sicer predvsem t.im. visoko sintakso, dočim srednjo in nižjo le omenja. Temu sledijo primeri iz konverzije podatkov, obdelave seznamov, hkratne obdelave programov (multitasking) ter primerjave strukturiranega programiranja s programskimi strukturami jezika PL/1. Knjiga je pisana strogo, natančno, poučno, skratka strokovno in razjasnjuje manj znana poglavja s področja PL/1 programiranja.

(21) A.Schulz, Einführung in das Programmieren in PL 1, Zal. Walter de Gruyter, Berlin, New York, 1975 (306 strani, mehkovezana, visok stavek).

Delo je uvod v PL/1 programiranje za bralca v nemškem jeziku.

INFORMATICA 2/1981

MALI RAZVOJNI SISTEM ZA MIKROPROCESOR 68000

M. ROGAČ,
D. HAFNER

UDK: 681.3-181.4

ISKRA ELEKTROMEHANIKA
TOZD RAČUNALNIKI KRANJ

Članek opisuje delovanje in uporabo malega razvojnega sistema, ki temelji na mikroprocesorju 68000. Sistem je pomemben za razvoj večjih mikroračunalniških sistemov, zasnovanih na 16-bitnem mikroprocesorju 68000.

Razvojni sistem je bil vključno s sistemskimi programi razvit v ISKRI, TOZD Računalniki Kranj.

SMALL DEVELOPMENT SYSTEM FOR MICROPROCESSOR 68000: This article describes operation and use of Small development system, based on microprocessor 68000. Small development system is very important for further development of microcomputer systems, based on 16-bit microprocessor 68000.

Development system and system software were developed in ISKRA, TOZD Računalniki Kranj.

1. UVOD

V svetu in tudi v Jugoslaviji je opazen prehod od 8-bitnih mikroprocesorjev na zmogljivejše 16-bitne. V ISKRI, TOZD Računalniki smo v mikroračunalniku ISKRADATA 1680 dosedaj uporobljali 8-bitni mikroprocesor 6800. Ker ima mikroprocesor 68000 podobne značilnosti in deluje podobno kot mikroprocesor 6800, smo se odločili za nadaljnji razvoj na osnovi mikroprocesorja 68000.

Razvojni sistem je namenjen za preizkušanje modulov, ki bodo vključeni v novi mikroračunalniški sistem ISKRADATA 90, vzporedno pa tudi za preizkušanje pripadajoče programske opreme. V ta namen je bil razvit tudi križni zbirnik za mikroprocesor 68000, ki teče na miniračunalniku ISKRADATA C-18.

2. ZGRADBA MALEGA RAZVOJNEGA SISTEMA

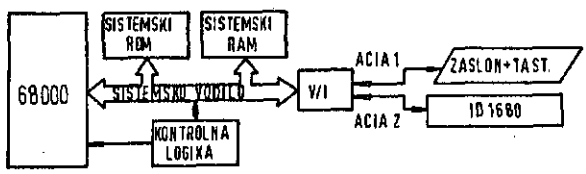
Mali razvojni sistem je izdelan na kartici, katere velikost je dvojna evropa. Strojna oprema razvojnega sistema /slika 1/ vsebuje:

-MIKROPROCESOR 68000 s 4 MHz sistemsko uro. Ko bodo dostopni zmogljivejši mikroprocesorji 68000, bo uporabljena 16 MHz sistemska ura.

-SISTEMSKI ROM: obsega 4K 16-bitnega pomnilniškega prostora in obsega lokacije med naslovi H'00000-H'01FFF. Sestavljen je iz dveh 8-bitnih EPROM-ov 2732, ki dajeta 16-bitno besedo. V sistemskem ROM-u je vpisan poleg nastavitve vektorjev tudi sistemski monitor.

-SISTEMSKI RAM: obsega 4K 16-bitnega pomnilniškega prostora na lokacijah H'20000-H'21FFF. Sestavljen je iz štirih RAM-ov 4016.

-SISTEMSKO VODILO: sistemsko vodilo je multipleksirano, kar pomeni manjše število signalov na vodilu, potrebna pa je dodatna krmilna logika. Naslovi in podatki so multipleksirani, operacije na vodilu so asinhrone.



Slika 1: Zgradba malega razvojnega sistema

-KONTROLNA LOGIKA: vsebuje logiko nepotrjenega prenosa - prekinitev "bus time-out". Če pride pri naslavljanju do naslova, ki sega izven naslovnega prostora, procesor ne dobi potrditve prenosa podatkov in čaka. Če po 0.1 ms ne dobi potrditve prenosa podatkov, kontrolna logika sproži logiko nepotrjenega prenosa, ki povzroči skok procesorja v prekinitveno rutino, preko katere sistem javi napako v naslavljanju in vzpostavi začetno stanje.

-V/I: vhodno/izhodno vezje vsebuje dve ACIA /asinhroni serijski komunikacijski vmesnik/. ACIA1 /na naslovu H'3FF01 oz. H'3FF03/ je namenjena za povezavo razvojnega sistema z zaslonom in tastaturo, ACIA2 /na naslovu H'3FF21 oz. H'3FF23/ pa je namenjena za vnos programov iz mikroračunalnika ID 1680 oz. miniračunalnika ID C-18.

3. SISTEMSKA PROGRAMSKA OPREMA

3.1 VNOS PROGRAMOV

Programe za mali razvojni sistem v zbirniku prevajamo s križnim zbirnikom, ki teče na miniračunalniku ID C-18. Prevedene programe preko serijskega izhoda (RS-232) pošljemo v mali razvojni sistem. Monitorski program za vnos podatkov v malem razvojnem sistemu pretvarja strojne kode zapisa S in jih shrani v RAM.

3.2 SISTEMSKI MONITOR

Sistemski monitor je zapisan v sistemskem ROM-u, vsebuje naslednje ukaze:

-DM XXXX YYYY Ukaz prikaže vsebino pomnilnika od naslova XXXX do naslova YYYY.

-SM XXXX LL MM NN ... Ukaz povzroči da na naslove od XXXX naprej shranjujemo vrednosti LL, MM, NN itd.

-GO XXXX Ukaz povzroči začetek izvajanja programa na lokaciji XXXX.

-T XXXX Ukaz povzroči izvajanje ukaza za ukazom od naslova XXXX dalje. Vrednosti registrov mikroprocesorja M68000 se po vsakem izvedenem ukazu izpišejo na zaslon. Izvajanje naslednjega ukaza omogočimo s pritiskom na eno od tipk tastature. Ta ukaz je krmiljen programsko in ne zahteva posebne strojne opreme.

-P XXXX Ukaz povzroči nalaganje podatkov iz ACIA2 v pomnilnik od naslova XXXX dalje.Program je zapisan v S formatu.

4. ZAKLJUČEK

Na malem razvojnem sistemu uspešno odkrivamo napake v programih in v križnem zbirniku. Preizkušamo tudi module za novi mikročunalniški sistem ISKRADATA 90.

5. LITERATURA

16-BIT MICROPROCESSOR, User's manual, Motorola 1979

MC68000 DESIGN MODULE, User's guide, Motorola 1980

MC68000 SYSTEMS CROSS MACRO ASSEMBLER, Reference manual, Motorola 1979

INFORMATICA 2/1981

PRIKAZ RAZVOJA OPERATIVNIH SISTEMA

VLADIMIR HOLODKOV

UDK: 681.3.065

ZOIL „VOJVODINA", RZ „VOJVODINA – AOP"
NOVI SAD

U radu je dat prikaz razvoja operativnih sistema računara na osnovu analize dostupne literature. Analizom je ustanovljeno da postoji pet osnovnih koncepcija operativnih sistema.

A SURVEY OF DEVELOPMENT OF THE COMPUTER OPERATING SYSTEMS

The paper gives the survey of development of the computer operating systems. It made the analysis of fifteen books. The analysis indicates that exists five types of operating systems.

1 UVOD

Jedna grupa autora pod pojmom operativnog sistema podrazumeva skup programskih modula unutar bilo kog računara koji upravlja resursima i reguliše rad hardvera. Pod resursima se podrazumevaju procesori, radna memorija, sekundarne memorije, ulazno/izlazni uredjaji, ulazno/izlazni kanali,datoteke, mreža za prenos podataka, terminali i sami podaci. Operativni sistem rešava konfliktne situacije metodama i tehnikama interrupta i optimizira izvodjenje svih programa računara i aplikacija korisnika. Programski moduli operativnog sistema predstavljaju sponu izmedju korisnika i gole mašine (hardvera). Ova spona je omogućena preko interpretatora jezika operativnog sistema (job control language, operator control language, administrator control language). Svaki od ovih jezika ima svoju pragmatiku, semantiku i sintaksu u mašinskom smislu tj. radi se o formalnim jezicima razumljivim mašini. Preko ovih jezika se mogu opisati: pravila dodele resursa i njihove razdele od strane više korisnika; redosled zahteva za resursima; koji moduli su nadležni za njihovo opsluženje; opis tipa resursa; nazivi programa; imena datoteka koje se zahtevaju itd. Ova grupa autora svrstava programske module operativnog sistema prema funkcijama koje obavljaju:

* upravljanje memorijom
* upravljanje procesorom
* upravljanje U/I uredjajima
* upravljanje informacijama.

Svaka od navedenih funkcija ima sledeće zadatke:

* da prati i pamti status resursa kojima upravlja
* da uskladjuje delovanje u smislu definisanja ko, kakve, kada i kako dobija resurse
* dodeljuje resurse na zahtev (od trenutka kada je posao kreiran od strane korisnika podvrgnut obradi tj. posao je u stanju "submit")
* oslobadja resurse na zahtev.

Druga grupa autora smatra da je operativni sistem računara skup upravljačkih programa sa tri osnovne funkcije:

* upravljanje računarom
* upravljanje poslom
* upravljanje podacima.

Treća grupa autora posmatra operativni sistem takodje kao skup upravljačkih programa grupisanih u tri dela:

* supervajzor je modul nadležan za upravljanje razmene podataka izmedju centralnog procesora i perifernih uredjaja
* monitor je modul zadužen za planiranje i raspoređivanje resursa u skladu sa zahtevima kreiranih poslova korisnika
* direktor je modul nadležan za koordinaciju ostalih modula operativnog sistema a u njegovoj nadležnosti je i interakcija sa operaterom na glavnoj konzoli računara. Po prirodi svog rada mora biti rezidentan u radnoj realnoj memoriji.

Paralelno razvoju tehnologije komponenti hardvera, razvijao se i operativni sistem u svojoj kompleksnosti, funkcionalnosti i obimu poslova koje je pokrivao. Biće prikazana istorija razvoja operativnih sistema po generacijama /14/:

Prva generacija (1955.-1962.) Jednostavni monitori poslova paketne obrade (batch) sa mogućnostima povezivanja više poslova u jedan niz (batch stream). Dizajniran za rad samo na jednom tipu i modelu računara.

Druga generacija (1960.-1968.) Jednostavni načini rada (omogućavao je paketnu obradu, obradu u realnom vremenu, obradu transakcija ili obradu u režimu razdeljenog vremena).Imali su ograničene sposobnosti multiprogramiranja, multiprocesiranja i upravljanja podacima. Dizajnirani za rad na jednom ili više modela računara.

Treća generacija (1965.-198?)Kompleksni višenamenski multiprogramski i multiprocesorski sistemi. Naglasak je pomeranje dizajna sa paketne na direktnu obradu. Dizajnirani za rad na jednoj familiji računara.

Četvrta generacija (1977.-19??)Modularno-slojeviti operativni sistemi napravljeni uz snažnu podršku firmvera. Većina funkcija je apsorbovana na nivou softverskih uslužnih programa. Pomeranje operativnih sistema orijentisanog procesoru na operativne sisteme orijentisane podacima (DB/DC koncept). Dizajnirani za rad u jednoj familiji računara sa više hardverskih generacija.

2 KLASIFIKACIJA OPERATIVNIH SISTEMA

Na osnovu analize dostupne literature, pokazalo se da u osnovi postoji pet struktura operativnog sistema prilagodjenih za režim rada računara:

* operativni sistem za rad u režimu razdeljenog vremena
* operativni sistem za rad u režimu realnog vremena
* operativni sistem za rad u režimu virtuelnih mašina
* operativni sistem za rad u režimu sa više procesora i
* operativni sistem za rad u režimu direktne obrade.

2.1 Operativni sistem za rad u režimu razdeljenog vremena (time-sharing)

Računari za rad u režimu razdeljenog vremena nude direktnu interakciju korisnika sa računarom simultanim pristupom centralnom procesoru. Time-sharing predstavlja metodu dodele vremena procesora nekolicini korisnika tako što se jedan vremenski ciklus procesora deli na intervale u kome jedan korisnik može raditi. Zahvaljujućitome, korisnik sistema dobija brzi odgovor na svoje zahteve.

U jednom trenutku vremena, u radnoj memoriji se može nalaziti više programa korisnika računara. Preostali programi se nalaze privremeno smešteni na sekundarnim memorijama. Svaki program u radnoj memoriji ima svoje vremenske intervale tako da u planiranom intervalu, procesor vrši obradu samo jednog programa. Procesor može preneti svoju pažnju na neki drugi program i pre isteka vremenskog intervala slučajevima prekida zbog U/I operacija. Način prenošenja kontrole procesora sa jednog programa na drugi je ovisan od strukture prioriteta posluženja i primenjenog algoritma. Dodelu vremenskog intervala u kome će procesor raditi nad jednim programom obavljaju programi operativnog sistema.

Neki od algoritama dodele prioriteta su:

* ciklički algoritam posluženja po kome se zahtevi korisnika računara smeštaju red čekanja
* modifikovani ciklički algoritam daje prioritet zahtevima sa kratkotrajnim vremenom obrade
* još veće favorizovanje zahteva sa kraćim vremenom obrade daje tzv. foreground background algoritam. U ovom algoritmu se uvodi više redova čekanja.
* može se nekim terminalima u konfiguraciji računara dodeliti odredjeni prioritet bez obzira na vreme obrade zahteva sa drugih terminala.

Rezoni za korišćenje računara za rad u režimu razdeljenog vremena su sledeći:

* maksimiziranje korišćenja resursa
* reduciranje vremena proteklog izmedju predaje zahteva za obradu procesoru i dobijanje rezultata za poslove manjeg obima (time turnaround)
* osposobljavanje udaljenih korisnika za unošenje podataka i prijem podataka iz računara preko komunikacione mreže
* da se udaljenim korisnicima daju prednosti u posedovanju računara, prilagodjavajući se njegovoj reakciji i U/I vremenskim intervalima potrebnim za izvodjenje programa
* da se zadovolje zahtevi okoline koja radi u realnom vremenu i da se omogući efikasna obrada zahteva (transakcija).

Moduli operativnog sistema u režimu razdeljenog vremena koji su odgovorni za upravljanje procesorom koriste pet različitih tehnika dodele vremenskog intervala procesora:

* sekvencijalna
* natural wait
* time slicing
* prioriteti
* po zahtevu (demand).

Analiza pokazuje da se računari za rad u režimu razdeljenog vremena koriste u dve kategorije:

* za paketnu obradu (batch processing)
* za daljinsku obradu (remote processing)
  * za Remote Job Entry rad
  * za konverzacioni način rada
  * za rad u realnom vremenu.

Druga klasifikacija operativnih sistema za rad u režimu razdeljenog vremena je:

* otvoreni operativni sistem, koji daje korisniku pristup svim mogućnostima (facility) operativnog sistema
* zatvoreni operativni sistem, koji limitiraju korisnika primenom specifičnih programskih jezika. Ovakvi operativni sistemi se koriste za konverzacioni način rada.

Sa aspekta rasporedjivanja vremena procesora i dodele procesora, operativni sistemi se razvrstavaju:

* sekvencijalno rasporedjivanje funkcija operativnog sistema sa podrškom perifernim uredjajima
* multiprogramski sistemi sa fiksnim ili varijabilnim brojem poslova i rasporedom prioriteta
* zatvoreni konverzacioni programski moduli operativnog sistema sa korišćenjem "time slicing" tehnike
* multiprogramski sistem sa jednim poslom visokog prioriteta ili postoji foreground particija za rad sistema iz prethodne tačke
* sistemi opšte namene za podršku otvorenom konverzacionom programiranju i podršku rada u paketnom režimu rada sa podrškom perifernim uredjajima.

Sa aspekta upravljanja memorijom susreću se sledeće tehnike:

* dodela cele memorije jednom poslu
* swapping korisnikovih particija
* segmentiranje stranica na zahtev
* prelistavanje (paging). Ova tehnika se javlja kod računara sa virtuelnom memorijom. Za efikasni rad, ova tehnika ima podršku sa dynamic address translation.

Sa aspekta upravljanja informacijama treba istaći da postoji potreba za razdelom sistemskih programa i njima pripadajućih podataka. Moguće rutine za razdelu na više korisnika su rutine iz domena upravljanja poslom i upravljanja podacima. Razdela modula omogućena je dodelom odgovarajućih atributa programu. Module sa atributom javnosti koristi nekoliko korisnika i oni njih mogu menjati samo u okviru svog programa. Module sa atributom privatnosti koristi korisnik koji poznaje zaštitni kod identifikacije programa.

2.2 Operativni sistem za rad u režimu realnog vremena (real-time)

Računar za rad u režimu realnog vremena mora biti sposoban da opsluži veliku količinu podataka, da ih analizira i daje odgovore na sve vremenski zavisne događaje. Računar mora biti sposoban da prima podatke onako brzo kako i dolaze jer u protivnom slučaju neki podaci mogu biti izgubljeni. Osnovna jedinica rada je transakcija.Pod transakcijom se podrazumeva skup logičk-h i u nizu povezanih parova poruka. Kod jednostavnih računara jedan par poruka obezbeđuje jednu transakciju. Pod parom poruka se podrazumeva skup podataka koji se prenose u računar i dobijanje odredjenog odgovora. Računari ovakvog tipa rade u većini slučajeva u uslovima postojanja komunikacione opreme.

Polazeći od pretpostavke da su bazni operativni sistemi nezavisni od režima u kojem računar deluje, treba naglasiti da organizacija programskih modula operativnog sistema za rad u režimu realnog vremena je takva da je modul za opsluživanje komunikacione opreme njegov sastavni deo.

Svaki od modula predstavlja interfejs sa korisnikovim aplikacionim rutinama. Ovaj interfejs se može omogućiti na dva načina:

* posebnim sogtverom koji je specijalizovan za rad u režimu realnog vremena a predstavlja podršku radu operativnog sistema
* poseban operativni sistem je dizajniran za rad u režimu realnog vremena a unutar sebe obuhvata module koji omogućavaju upravljanje paketnom obradom i module koji omogućavaju rad u realnom vremenu.

Zbog povećanja propusne moći računara, operativni sistem dozvoljava i multiprogramski rad. To ipak ne znači da se svaki poseban program izvršava brže već naprotiv.

Osnovne karakteristike operativnog sistema za rad u režimu realnog vremena su:

* obrada transakcija
* programi su rezidentni u radnoj memoriji duži vremenski period (zbog potrebe korisnika)
* simultano procesiranje poruka (koriste se tehnike duple baferizacije, multiprogramiranja i multi-threadinga)
* upotreba tehnike "tasking" tako da se poruke i odgovori smeštaju u redove čekanja sa definisanim prioritetima.

Struktura operativnog sistema za rad u režimu realnog vremena je data na slci 1. Spoljni izgled ukazuje na aplikativni karakter operativnog sistema jer se koristi uz standardni bazni operativni sistem. Prikazana struktura omogućava zadovoljenje zahteva za perifernim prenosom a kontroliše i multi-threading deljenjem vremenskog ciklusa procesora za potrebe raznih poruka koje stižu u računar. Omogućeno je i preklapanje rada procesora sa operacijama perifernog transfera (overlaying). Pošto se upravljanje u takvoj strukturi operativnog sistema vrši bez zahteva za resursima osnovnog mašinskog operativnog sistema, ovakav aplikativni karakter daje efikasniji rad u odnosu na korišćenje vremena procesora.

Korisnikovi aplikacioni programi prihvataju poruke koje su obradjene od strane sequence monitora i ako se na primer zahteva čitanje sloga tada rutina prosledjuje ovaj zahtev preko request monitora. Dakle, svi korisnikovi aplikacioni moduli se pozivaju iz jednog standardnog interfejsa (sequence monitora) a prosledjuje sve zahteve na drugi standardni interfejs (request monitor). Sa ovakvom strukturom je moguće različite poruke servisirati u istom trenutku sa različitim rutinama za opsluživanje (handling). Važna komponenta prikazane strukture operativnog sistema na slici 1 je message block.

Prava struktura message bloka je ovisna od dizajna korisnikove aplikacione rutine. Ovi blokovi su pod kontrolom communications handlera i u njemu su rezidentni.

Konkretne funkcije handlera se dinamički menjaju sa tipom aplikacione rutine.

Vidljivo je sa slike 1 da operativni sistem ima tri monitora sa sledećim funkcijama:

* request monitor
  - ispituje zahteve u request area message bloka
  - ako je prisutan zahtev za nekom drugom aplikacionom rutinom od one nad kojom trenutno ide obrada, onda request monitor smešta adresu message bloka u red čekanja sequence monitora
  - ako postoji zahtev za nekim od handlera, onda odlučuje koji je i smešta adresu message bloka u odgovarajuću ulaznu zonu selektovanog handlera
  - predaje kontrolu service monitoru.

* service monitor

  Service monitor po primitku kontrole od request monitora proziva specificiranim redosledom handlere. Posle poziva handlera vrši se predaja kontrole sequence monitoru. Ako sequence monitor odmah vraća kontrolu na njega onda to znači da ne postoji message blok koji čeka na neku aplikacionu rutinu. U tim trenucima nema više obrade zahteva. Tada service monitor predaje kontrolu osnovnom mašinskom operativnom sistemu izvodjenjem tzv. "multiple wait" instrukcije. Ovakav zastoj se odnosi na sve događaje koji se tiču particije memorije namenjene za rad u realnom vremenu. Ovim se želi izbeći da service monitor analizira nove događaje npr. prijem nove poruke.

  Kada jedan od događaja čeka da se pojavi u sistemu, tada operativni sistem vraća kontrolu na sledeću instrukciju iza "wait" komande te service monitor ponavlja svoj ciklus: poziv handlera i predaja kontrole sequence monitoru.

  Service monitor ima zadatak da prikuplja u "log" datoteku sve greške koje se jave da bi se sprečilo automatsko poništavanje prisutne aplikacione rutine od strane operativnog sistema. Ukoliko je u radu bio neki od handlera ili drugi monitori tada u slučaju pojave bilo kakve greške se javlja prekid koji ukazuje operativnom sistemu da prenese kontrolu na

service monitor. Tada je service monitor u poziciji da vrši analizu greške.

Nakon analize, service monitor informiše korisnika terminala preko komunikacionog softvera o tipu greške i eventualno o tome kako da se oporavi od nje. U tu svrhu on generiše message blok koga communications handler dalje prosledjuje zahvaljujući posebnoj sistemskoj rutini koju poziva preko sequence monitora.

* sequence monitor

  Sequnce monitor čuva tabelu koja sadrži adrese ulaska u svaku aplikacionu rutinu i adrese redova čekanja message blokova koji čekaju na svoje aplikacione rutine. Ove adrese su u tabeli sortirane po prioritetima u message area message bloka. Zahvaljujući tome, sequence monitor vrši selekciju poruka za obradu. Sequence monitor sadrži i tabelu naziva modula sa njihovim statusima. Ovi statusi su bitni ako se pojavi preklapanje (overlaying). Tada na scenu stupa overlaying hanlder koji preko statusa aplikacionih rutina definiše startne adrese rutina u radnoj memoriji i adresu preklapanja.

2.3 Operativni sistem za rad u režimu virtuelnih mašina

Pored interpretatora jezika operativnog sistema, prisutni su kod nekih računara i interpretatori jezika virtuelnih mašina (virtual machine language). To je skup mašinskih instrukcija koje stoje korisniku na raspolaganju ali samo u toku izvodjenja programa. Za olakšano upravljanje fizičkom mašinom i izolaciju korisnika jednog od drugog, operativni sistem za rad u režimu virtuelnih mašina mora sprečiti korisnikove programe da izvode one instrukcije koje bi izazvale uništavanje procedura i podataka unutar operativnog sistema. Skup instrukcija iz "virtual machine language" je podskup originalnog mašinskog jezika ali se stvara iluzija samostalnosti rada korisnika kao da mu pripada ceo računar.

Neka u računskom centru postoji odredjena konfiguracija fuzičkog računara. Neka je takav računar kao proširena mašina /9/ snabdeven sa operativnim sistemom uslovno nazvan "Monitor Virtuelne Mašine" (MVM). Neka je struktura ovakvog operativnog sistema takva da je omogućeno multipleksiranje samo fizičkih resursa od strane više korisnika i da MVM ne nudi više nikakve druge funkcionalne pogodnosti. Pod ovakvim uslovima MVM omogućava da se jedan stvaran računar pojavljuje kao više zasebnih računara. Pored ovoga, MVM stvara utisak postojanja više "golih" mašina te svaki korisnik bira svoj operativni sistem koji će se izvoditi na njegovom privatiziranom računaru. Na slici 2 je data organizacija virtuelnih mašina koja će poslužiti za objašnjenje mehanizma njihovog rada.

Sa slike 2 je vidljivo da se pod MVM izvode programi operativnih sistema npr. OS1, OS2 i OS3. Programi ovih operativnih sistema se nalaze u "problem state" (stanje zadatka tj. izvodjenja). Ali, oni se ponašaju kao da su u "supervisor state". To znači, kada se izvode privilegovane instrukcije OS(i) operativnih sistema tada se generiše prekid koji ukazuje na predaju kontrole na MVM operativni sistem i on tada izvodi privilegovane komande.

Prekidi, generisani zbog izvodjenja privilegovanih komandi predstavljaju interfejs izmedju korisnika i MVM operativnog sistema i ovim prekidima se ne remeti rad programa korisnika.

Operativni sistemi za rad u režimu virtuelnih mašina se sastoji od dve komponente:

* upravljački program MVM koji izvodi funkcije multipleksiranja procesora, U/I uredjaja i memorije da bi se proizvele virtuelne mašine
* sistem za vodjenje konverzacije sa virtuelnom mašinom (conversational monitor systems - CMS) koji izvršava funkcije: obrade komandi, upravljanje informacijama za potrebe rada računara i ograničenog upravljanja U/I uredjajima.

Multipleksiranje fizičkog računara je dovelo do toga da komponenta CMS operativnog sistema koristi memoriju dodeljenu virtuelnoj mašini kao da je to cela neprekidna memorija (single contigious allocation) za izvodjenje samo jednog korisnikovog programa. Ova činjenica ukazuje na primenu jednostavnih tehnika upravljanja memorijom i procesorom.

Upravljački program MVM u zajednici sa mogućnostima hardvera simulira da svaka virtuelna mašina ima procesor, memoriju, U/I uredjaje, operatersku konzolu i panel za nadgledanje rada sistema. Pored ovoga, MVM omogućava ažurnost odredjenog skupa tabela koje sadrže opis i status svih komponenti virtuelne mašine. Preko ovih tabela se uspostavlja veza izmedju komponenti fizičkog računara i virtuelne mašine. Iako svaka virtuelna mašina ima svoj operativni sistem sa pratećim softverom, bilo bi ne efikasno da se svakoj virtuelnoj mašini ponavljaju zajednički skupovi programa operativnih sistema. Može se obezbediti posebno područje u kojoj će se vršiti razdeljivanje zajedničkih programa i podataka pri čemu je osnovna jedinica razdeljivanja segmenat koji ima varijabilnu dužinu. Razdeljeni segmenti mogu biti ili deo aplikacionog programa ili deo operativnog sistema. Pri tome, svaki segmenat se sastoji od jednog ili više programa u mašinskom obliku spremnom za izvodjenje ili se sastoji od jednog ili više područja sa podacima potrebnim za rad programa operativnog sistema virtuelne mašine OS(i).

Jedna od prednosti segmentiranja je da se povećava efikasnost korišćenja memorije i propusna moć računara.

Kod jednog fizičkog računara, na kome se može izvoditi virtuelizacija mašine, razlika izmedju "problem state" i "supervisor state" izvodjenja komandi je omogućila da MVM izvodi većinu instrukcija virtuelnih mašina. Kada se procesor nalazi u "problem state" radu, bilo kakav pokušaj izvodjenja instrukcija koja menja ili proverava stanje celog računara izaziva programski prekid. Prema tome, izvodjenjem komandi virtuelnih mašina dok se nalazi u "problem state" modu, MVM je obezbedjen od gubljenja kontrole. Kontrola se gubi nailaskom privilegovane komande te u tom slučaju fizički računar simulira odgovarajući funkcionalni efekat privilegovane komande tako što iz tabele opisa virtuelnog procesora, čiji je status posebno definisan, gleda da li je virtuelna mašina sa izazvanom privilegovanom komandom u "problem state" ili u "supervisor state" modu. Ako je virtuelna mašina bila u "problem state" modu, MVM će simulirati programski prekid na virtuelnoj mašini. Ako je virtuelna mašina bila u "supervisor state" modu, MVM će morati da dešifruje komandu i izvršiće simulaciju te komande.

Sve U/I operacije su simulirane. Kada virtuelna mašina pokuša da izvede U/I operaciju javlja se prekid koji izaziva da se kontrola prenosi na MVM. Moguće su tri situacije:

* ako zahtevani uredjaj fizički postoji (ovo se proverava zbog prisutne opasnosti da se ne ažurira tabela statusa hardvera virtuelne mašine od strane administratora sistema) i pridružen je na virtuelnu mašinu koja je generisala prekid tada se U/I operacija odmah izvodi

* ako postoji sličan uredjaj tada se odredjenim algoritmom modifikuje U/I komanda i nakon toga se izvodi (npr. razdela jednog diska na više mini diskova)

* neki uredjaji su simulirani dodatnom tehnikom SPOOL-inga.

Pokazalo se da je najpodesnija tehnika upravljanja memorijom "prelistavanje na zahtev" (demand paging) i svakoj virtuelnoj mašini se obezbedjuje sopstvena virtuelna memorija.

Komponenta operativnog sistema rad u režimu realnog vremena MVM ne obezbedjuje upravljanje informacijama. Takva mogućnost je pod kontrolom konkretnih operativnih sistema virtuelne mašine OS(i). To znači, upravljanje informacijama je u nadležnosti komponente CMS koja je dizajnirana za rad u okolini (environment) virtuelne mašine. CMS opslužuje poseban katalog datoteka za svaki uredjaj sa direktnim pristupom (sekundarne memorije). CMS komponenta radi sa dva takva uredjaja: jedan se koristi za držanje nerezidentnih delova CMS (sistemski disk) a drugi uredjaj se koristi za memorisanje korisnikovih stalnih datoteka. MVM komponenta dopušta da dva ili više virtuelnih uredjaja sekundarne memorije bude mapirano na istom fizičkom uredjaju te tada postoji potreba da u fizičkom računaru egzistira samo jedan stvaran CMS sistemski disk koga će razdeliti sve CMS komponente svih operativnih sistema OS(i). MVM onemogućava korisnika da u toku rada vrši modifikaciju sistemskog diska.

2.4 Operativni sistem za rad u režimu sa više procesora

Procesor je fizička komponenta računara koji sekvencijalno sprovodi obradu definisanu programom. Više procesora može konkurentno da radi pri čemu se razdeljuje zajednička memorija po jednom algoritmu zajedničkom za sve procesore prisutne u konfiguraciji. U računaru sa više procesora jedan se može proglasiti za centralni (onaj koji radi sa podacima u radnoj memoriji) a ostali su terminali ili periferni uredjaji (kanali; koji prenose podatke iz okoline računara u radnu memoriju).

U osnovi, postoji tri organizacije i načina rada operativnih sistema u režimu sa više procesora /5/:

* master-slave
* separatno izvodjenje operativnog sistema za svaki procesor
* simetrični ili anonimni tretman svih procesora.

*. Master-slave tehnika

Kod ove tehnike, jedan procesor (master - gospodar) rukovodi statusom svih obrada u računaru i rasporedjuje ih na druge procesore u konfiguraciji (slave - rob). Na primer, master procesor bira obradu koja će se izvoditi, pronalazi raspoloživi procesor i izvodi instrukciju "Start processor". Slave procesor startuje izvodjenje obrade počev od specificirane adrese lokacije u radnoj memoriji. Kada slave procesor naidje na eki izuzetan dogadjaj kao U/I prekidi zbog zahteva za odredjenim perifernim uredjajem, onda on generiše prekid i kontrola se prenosi na master procesor a slave procesor stopira sve svoje aktivnosti. Celokupan rad operativnog sistema u ovakvim uslovima je nedemokratski.

* Separatno izvodjenje operativnog sistema za svaki procesor

Kako se radna memorija razdeljuje na više procesora, ne postoji potreba za potpuno separatnim kopijama rutina operativnog sistema za svaki procesor. Ali kod ovog tipa organizacije rada operativnog sistema za rad u režimu sa više procesora se javlja situacija da se na svakom procesoru autonomno izvode njegove izvršne supervajzorske i support funkcije kao da se radi o monoprocesoru. Svaki posao koji ulazi u računar se dodeljuje jednom procesoru i on se izvodi i obradjuje na njemu do potpunog okončanja.

* Simetrični ili autonomni tretman procesora

Demokratskiji pristup radu računara sa više procesora je da se svi procesori tretiraju ekvivalentno. Pod takvim uslovima, lista svih poslova u računaru i njihov status je zapamćen u jednom području memorije što omogućava da bilo koji procesor ima pristup takvoj listi. Svaki procesor koristi isti algoritam rasporedjivanja poslova. Grupa identičnih procesora razdeljuje zajedničku radnu memoriju, U/I kanale i U/I uredjaje.

Na osnovu ove tri tehnikeorganizovanja operativnog sistema za rad u režimu sa više procesora, tendencija je da više procesora razdeljuje jednu zajedničku radnu memoriju. Izračunavanja se odvijaju na različitim procesorima koji vrše obradu jednog posla ali svaki od njih može prilaziti istim segmentima memorije. Posledica ovoga je da se zahteva dinamička relokacija dela radne memorije. U tu svrhu su namenjeni zasebni bazni registri za memorisanje baznih adresa a njima mogu prići svi procesori. Dakle, upravljanje memorijom kod računara sa više procesora koristi tehniku segmentiranja memorije. Jedan od uzroka koji može dovesti do konfliktne situacije "deadlock" (mrtva tačka sistema) je da se dva različita procesora blokiraju u približno istom trenutku i nakon memorisanja statusa tekućih poslova traže nov posao. Kako su procesori medjusobno neovisni a koriste isti algoritam rasporedjivanja poslova, može se desiti da zatraže obradu istog posla. Ovakva situacija je razrešena jednom dopunom algoritma tako da svaki procesor proverava status specijalnog "lock" bita u tabeli poslova koji su spremni za izvodjenje. Procesor testira takav bit i ako je na primer postavljen na vrednost "1" ne može da opsluži zatraženi posao sve dotle dok ga procesor koji radi nad takvim poslom ne kompletira. Nakon kompletiranja posla, "lock" bit se automatski postavlja na vrednost "0" od strane procesora koji je servisirao taj posao.

Važan faktor koji odredjuje strukturu računara sa više procesora jeste način povezivanja procesora odnosno izmena podataka izmedju njih. Praktikuju se četri načina povezivanja /1/:

* povezivanje preko zajedničkog perifernog uredjaja
* povezivanje preko zajedničke memorije (radne i sekundarne)
* povezivanje preko specijalnih uredjaja (komutatora; komutator povezuje procesore sa delovima memorije postupkom time-sharinga; koordinator

kanala; to je procesor za kontrolu U/I aktivnosti)
* neposredno sprezanje procesora preko tzv. magistrala.

Za razmenu podataka izmedju procesora su razvijene tri osnovne tehnike:
* prostorna selekcija (komutatori)
* kodna selekcija
- vremenska selekcija (vremensko multipleksiranje.

U najnovije vreme se ide na decentralizaciju funkcija operativnog sistema u računaru sa više procesora. To se ostvaruje specijalizovanim funkcionim procesorima izradjenim u obliku firmvera. Medjuveza okoline računara i resursa računara je operativni sistem a opsluživanje okoline je omogućeno preko interfejsa izvedenog u obliku funkcionog procesora (slika 3).

Iz razloga sigurnosti i izbegavanja konfliktnih situacija kao što su blokada računara, obezbedjeno je da upravljanje datotekama (pristup sistemskim datotekama i tabelama) bude neposredno pod kontrolom operativnog sistema.

2.5 Operativni sistem za rad u režimu direktne obrade (on-line) (Data Base / Data Communications System)

Rad u režimu direktne obrade predstavlja takav rad na računaru gde se izvršavaju obrade svih zahteva poslatih sa mesta gde su nastali podaci za obradu i dobijanje trenutnog odgovora o izvršenoj obradi (upis novih slogova, ažuriranje postojećih ili brisanje starih slogova). Ukoliko bi operativni sistem u ovakvim uslovima sam obavljao nadzor i upravljanje bio bi previše kompleksan, spor i zauzimao bi veliki deo korisnog prostora radne realne memorije. Zbog ovoga, osnovnom operativnom sistemu su dodata dva softverski izvedene modula:

* sistem za upravljanje prenosom podataka (data communications software)

* sistem za upravljanje bazama podataka (data base management system)

Pored zahteva obrade korisnikovih podataka, operativni sistem za rad u režimu direktne obrade mora pokrivati zahteve interaktinog programiranja i paketne (batch) obrade.

* Sistem za upravljanje prenosom podataka

Za olakšani interfejs izmedju centralnog procesora i komunikacione mreže, nude se komunikacioni kontroleri kao medjuveza kanala centralnog procesora i linijske opreme (prenosne veze, modemi, terminali).

Osnovne funkcije su:
* kodna konverzija
* obezbedjenje odgovarajućih fizičkih uslova za rad modema
* prikupljanje serijskih bitova sa linije i njihova transformacija u oblik karaktera
* prenos i prijem znakova odredjenom standardnom brzinom
* prepoznavanje primljenih i generisanje znakova koji služe za sinhronizaciju ili za uobličavanje karaktera
* provera pariteta na nivou znaka i na nivou bloka
* opsluživanje korisnika i centralnog procesora o greškama u toku rada
* prepoznavanje upravljačkih znakova linijskog protokola (data link control characters)
* prozivanje terminala (polling)
* opsluživanje više korisnika na jednoj liniji (multi point veza)
* precizno odbrojavanje takt impulsa pri prijemu i prenosu je omogućeno elektronskom logikom pridružene svakom interfejsu prema komunikacionoj liniji.

Komunikacioni kontroleri su klasifikovani prema izvedbi proizvodjača /13/:

* singl linijski kontroleri
* multi-linijski hardverski kontroleri (skaneri; multipleksori)
* multi-linijski softverski kontroleri (procesori na prednjem kraju; front-end-procesori).

Na slici 4 je dat odnos komunikacionog softvera prema jednom računaru za direktnu obradu.

Komunikacioni softver ima još i sledeće zadatke:
* automatsko vodjenje dijaloga po odredjenim linijskim protokolima (BSC, SDLC, HDLC, X.25 itd.)
* kontrola mreže konsultovanjem tabela hardverskih i simboličkih adresa udaljenih uredjaja unutar adrese linije
* baferizacija poruka
* komutiranje poruka (message switching)
* omogućava pisanje, testiranje i izvodjenje više programa sa udaljenih terminala
* detektovanje grešaka. Ako se ponavaljanjem transmisije podataka ne dodje do željenog efekta, komunikacioni softver (ako ima takvu mogućnost) može izolovati korak po korak sve učesnike u komunikaciji (modem na mestu računara, linija, udaljeni modem i terminal) i utvrditi koji je od njih izazvao grešku u sistemu za prenos podataka. Izveštaji se memorišu u posebnom redu čekanja (queue) i po želji se mogu prikazati na nekom perifernom uredjaju
* dozvoljava re-konfiguraciju računara, izmenom unapred pripremljenih tabela hardverskih adresa terminala i linija a na zahtev operatera. Ovo je često potrebno kod pojave neispravnog elementa u mreži za komunikaciju pa se želi preći na back-up. Druga potreba je kada su korisnici računara rasporedili svoj rad u smenama pa se u jednoj smeni radi sa jednom konfiguracijom mreže za komunikaciju a u drugoj smeni sa drugom mrežom
* predstavlja interfejs izmedju aplikacionog programa i resursa računara.

* Sistem za upravljanje bazama podataka (DBMS)

U radu sa softverom za upravljanje bazama podataka postoji niz dogadjaja kod zahteva aplikacionog programa (čitanje, pisanje ili ažuriranje slogova baze podataka) (slika 5).

Operativni sistem preuzima kontrolu nakon zahteva aplikacionog programa za ažuriranjem, pisanjem ili čitanjem slogova baze podataka i radi sa uredjajima sekundarne memorije gde su zapamćeni ili gde će se zapamtiti podaci. Zahtevani podaci (ako se radi o čitanju ili ažuriranju) se prenose izmedju sekundarne memorije i sistemskih bafera.

Procedure pretraživanja sloga u bazi podataka i procedure adresiranja su sastavni deo operativnog sistema.

Procedure za zaštitu i proveru identiteta podataka pomoću odredjenih lozinki poznatih operativnom sistemu su sastavni deo operativnog sistema.

Procedure automatskog restartovanja i oporavljanja od grešaka (recovery) su sastavni deo operativnog sistema.

Operativni sistem za rad u režimu direktne obrade mora da omogući sledeće:

* pristup do baze podataka (preko DBMS)
* pristup do udaljenih terminala preko odgovarajućeg komunikacionog softvera
* aplikacione funkcije
  - pristup do podataka
  - ažuriranje podataka
  - analiza podataka
  - priprema izveštaja
* automatsko restartovanje i recovery računara.

Jedna od mogućih struktura ovakvog operativnog sistema je data na slici 6.

Da bi se omogućio rad u režimu direktne obrade biće faktografski navedene komponente komunikacionog softvera koje još nisu spomenute a prikazane su na slici 6. To su sledeće funkcije:

* terminalska funkcija
  - Telecommunications Access Method
  - Terminal Control Program
  - Basic Mapping Support
* kontrolne funkcije
  - Program Control
  - Interval Control
  - Task Control
  - Storage Control
  - Temporary Data Control

Više detalja ima u /15/.

Tok aplikacionog programa je:

1 - čitanje koda transakcije i ulaznih podataka
2 - provera koda i zahtev za memorijom
3 - izbor aplikacionog programa na osnovu zahteva
4 - punjenje programa u prethodno odredjeni deo memorije
5 - prijem korisnikovog zahteva
6 - pristup do baze podataka
7 - formatiziranje i ispis izlaza

6 ZAKLJUČAK

U radu se faktografski navode strukture i funkcije pojedinih modula operativnih sistema. U analizi je razvoj operativnih sistema posmatran transparentno u odnosu na proizvodjača odnosno modele i tipove računara.

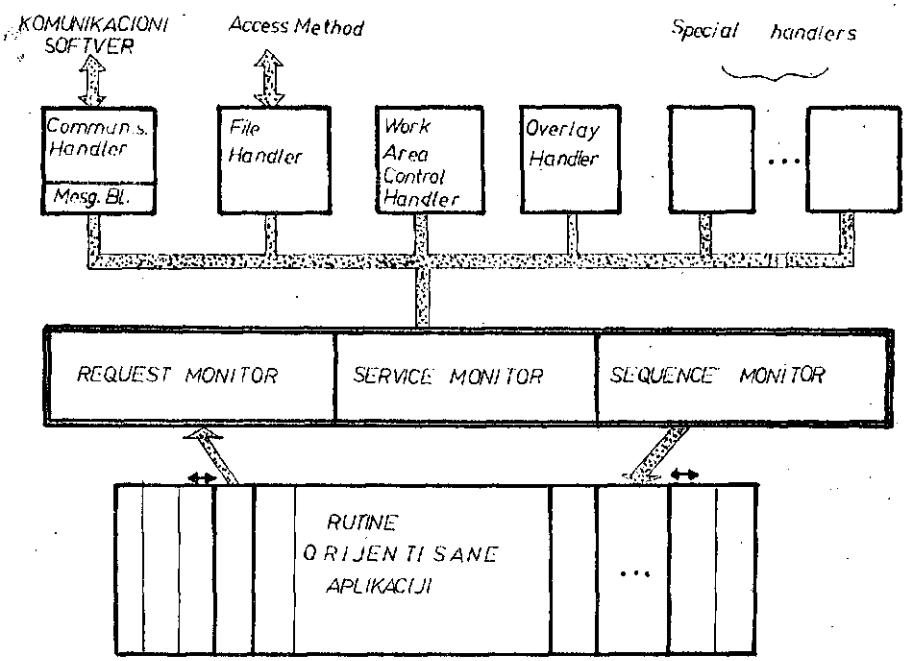
Nije bilo reči o prednostima i nedostacima pojedinih operativnih sistema jer je svaki tako dizajniran i balansiran da korisnik računara u svom radu ne oseća nedostatke primenjenih tehnika i algoritama jer je pretpostavka da korisnik pri nabavci računara vrši izbor i tehno-ekonomsko vrednovanje računara.

Analizom su obuhvaćeni operativni sistemi "velikih" računara i do danas bitnih novosti u razvoju operativnih sistema u odnosu na pet navedenih koncepcija nema. To se može videti i iz godišnjeg prikaza softverskih paketa za 1980. godinu u članku "Evaluating off-the-shelf programs" u časopisu "Datamation" od decembra 1980. godine u kome je data ocena 2.373 korisnika o 114 raznih softverskih produkata (pp. 85-120).

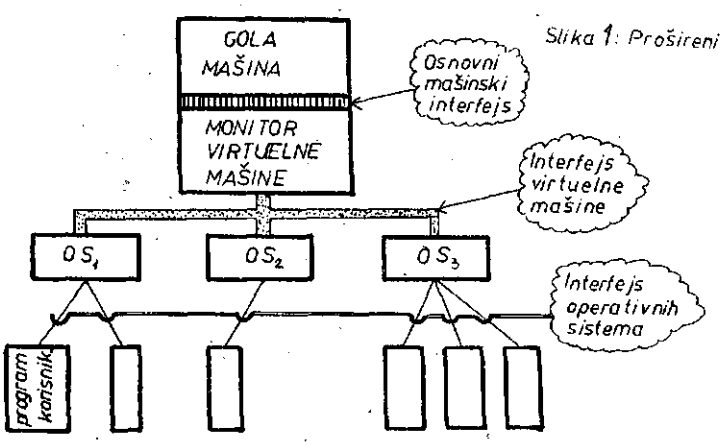
Stoji činjenica da kod današnjih računara je teško praviti kategorizaciju operativnih sistema jer većina od njih podržava u menjoj-većoj meri pet koncepcija navedenih u ovom radu.

Bibliografija

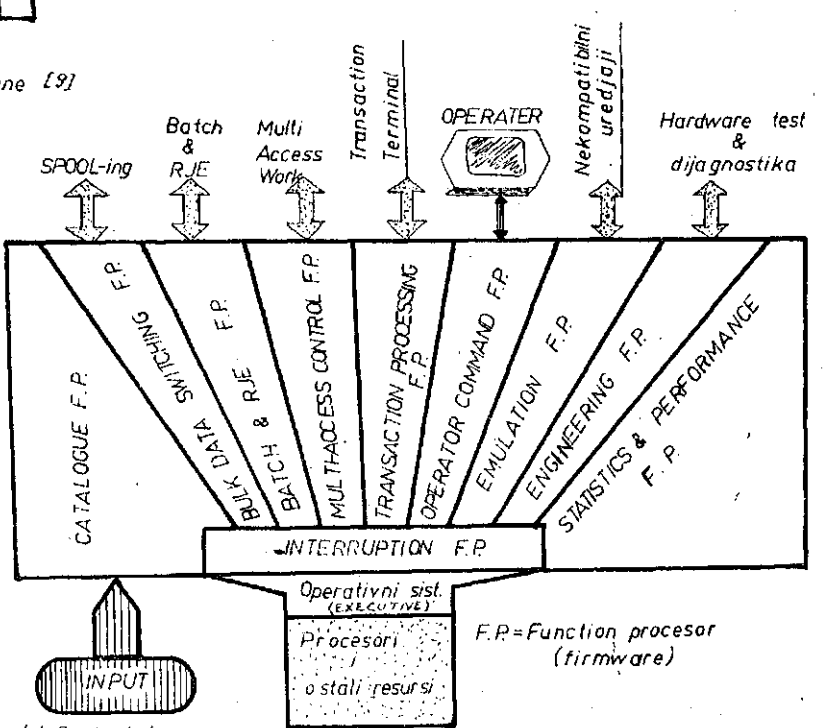
1. Aleksić T.
   SISTEMI ZA OBRADU PODATAKA, II dop.izdanje
   Skripta Elektrotehničkog fakulteta u Beogradu
   Beograd, 1977.
2. Broner Yu. D.
   MATEMATIČESKOE OBESPEČENIE SISTEM S RAZDELENIEM VREMENI
   Statistika
   Moskva, 1976.
3. Cohen L. J.
   OPERATING SYSTEM - Analysis and Design
   Spartan Book Co.
   New York (prevod na ruski, 1975.)
4. Donovan J. J.
   SYSTEMS PROGRAMMING
   McGraw Hill Kogakusha Ltd.
   Tokyo, 1972.
5. Enslow P. H.
   MULTIPROCESSORS AND PARALELL PROCESSING
   John Wiley & Sons
   New York, 1974.
6. Hansen P. B.
   OPERATING SYSTEM PRINCIPLES
   Prentice Hall
   New York, 1973.
7. Katzan H.
   ADVANCED PROGRAMMING - Programming and Operating Systems
   Van Nostrand Reinhold Co.
   New York, 1970.
8. Katzan H.
   OPERATING SYSTEMS - A pragmatic approach
   Van Nostrand Reinhold Co.
   New York, 1973. (prevod na ruski, 1976.)
9. Madnick S.E., Donovan J.J
   OPERATING SYSTEMS
   McGRaw Hill Book Co.
   New York, 1974.
10. Martin J.
    COMPUTER DATA-BASE ORGANIZATION,sec.edit.
    Prenitice Hall Inc.,New York,1977.
11. Martin J.
    SYSTEMS ANALYSIS FOR DATA TRANSMISSION
    Prentice Hall Inc.,New York,1972. (prevod na ruski, 1975.)
12. Paškeev S.D.
    OSNOVY MUL´TIPROGRAMMIROVANIJA DLJA SPECIALIZIROVANNYH VYČISLITEL´NYH SISTEM
    Sovetskoe radio, Moskva, 1972.
13. Tebs D., Collins G.
    REAL-TIME SYSTEMS - Management and Design
    McGraw Hill Book Co. (UK), London, 1977.
14. Weizer N.
    A HISTORY OF OPERATING SYSTEMS
    DATAMATION, januar 1981.
15. ***
    IBM CICS/VS Application Programmer Reference Manual (Command Level)
    SC - 33-0077



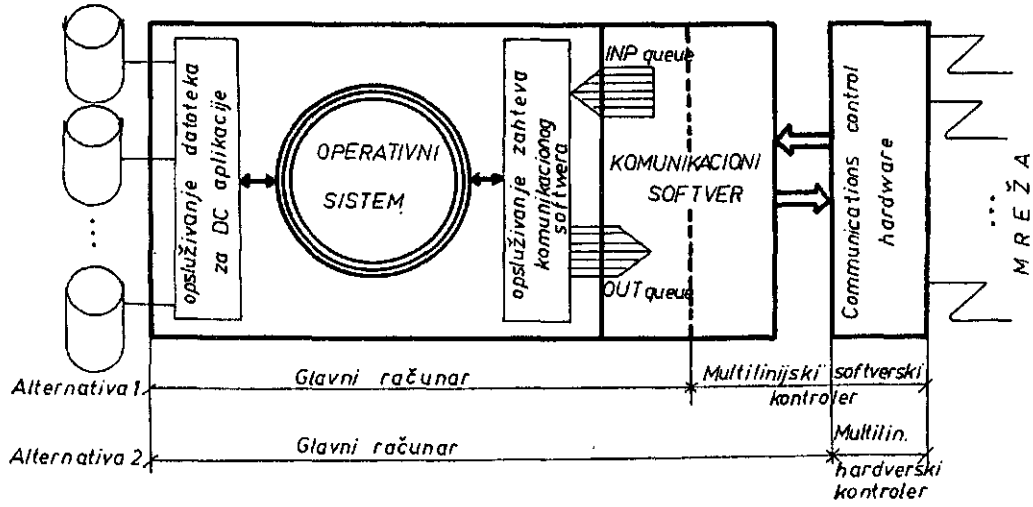
Slika 1: Proširen operativni sistem za rad u režimu realnog vremena [13]



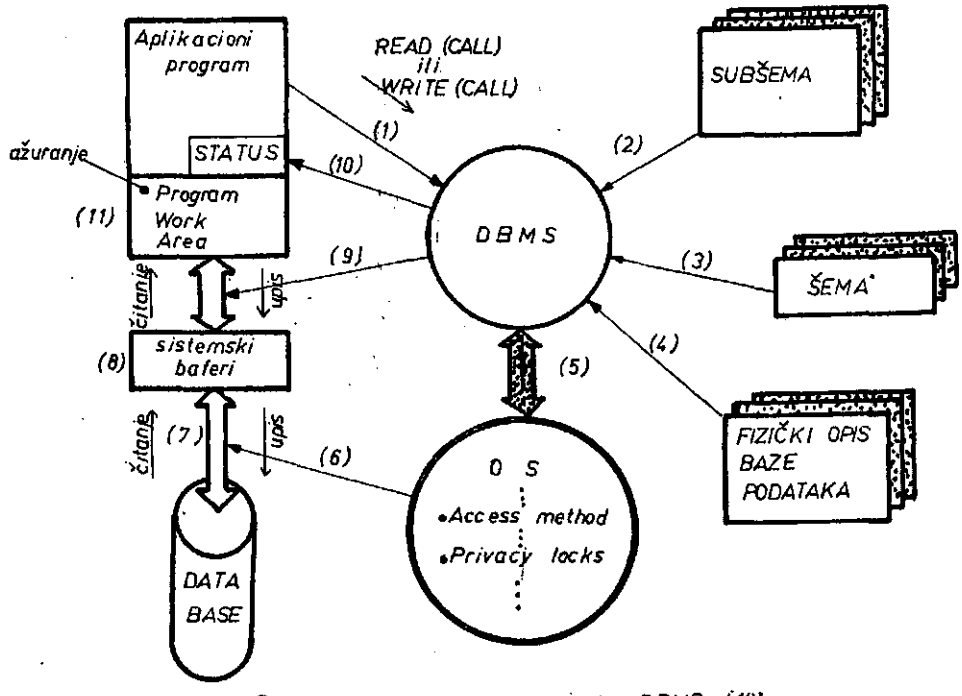
Slika 2: Programska organizacija virtuelne mašine [9]



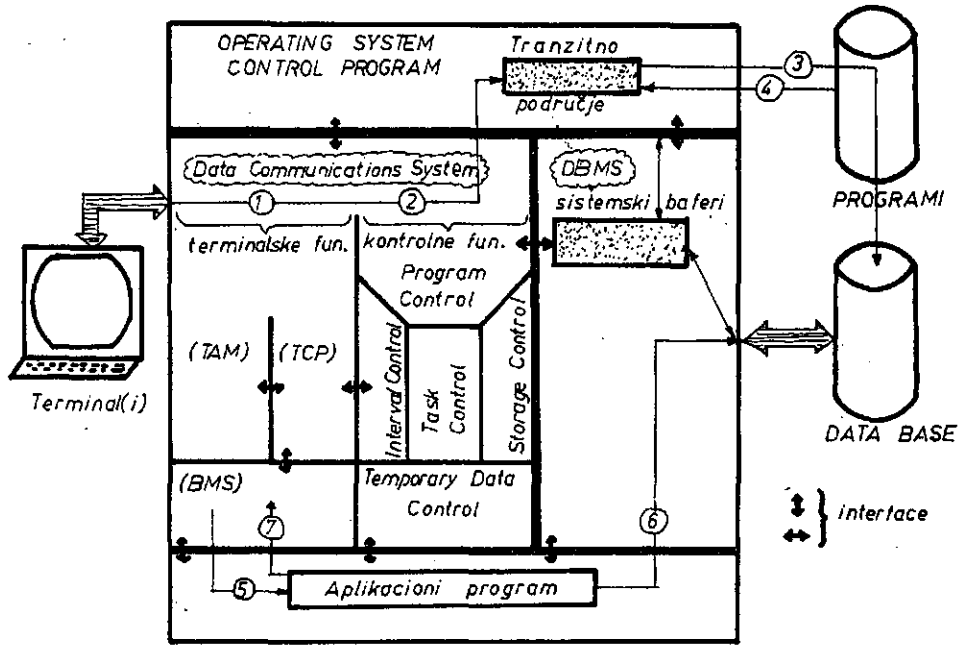
Slika 3: Firmwarsko upravljanje računarom sa više procesora (VME/K ICL 2900)



Slika 4: Veza komunikacionog softvera i operativnog sistema [13]



Slika 5: Redosled događaja u radu DBMS [10]



Slika 6: Programska organizacija operativnog sistema za rad u režimu direktne obrade [15]

INFORMATICA 2/1981

PRISTOP K REŠITVI MEDRAČUNALNIŠKE KOMUNIKACIJE V SISTEMIH Z DISTRIBUIRANIM KRMILJENJEM

ŽARKO NOVAKOVIĆ,
BRANISLAV POPOVIĆ

UDK: 681.3:621.398

ISKRA TELEKOMUNIKACIJE, KRANJ

V referatu podajamo eno od rešitev za komunikacijo med računalniki večračunalniškega sistema. Rešitev je uporabljena v sistemu telefonskih central ISKRA 2000. Rešitev temelji na zlogovnem protokolu s potrjevanjem sprejema zlogov in je primerna za zvezdasto konfiguracijo računalniške mreže.

AN APPROACH TOWARD THE SOLUTION OF COMPUTER COMMUNICATIONS IN SYSTEMS WITH DISTRIBUTED CONTROL: The paper describes a solution to the problem of communication between computers in a multicomputer system. The solution is applied within ISKRA 2000 - a system based on a byte oriented protocol with acknowledgement of individual bytes and applicable in star computer networks.

1. UVOD

Sistem ISKRA 2000 je modularni in digitalni sistem telefonskih central. Z modulom pojmujemo skupek telekomunikacijskih linij (terminalov) in je krmiljen z lastnim procesorjem. Moduli se povezujejo v sistem s pomočjo skupinskega stikala, preko katerega se povezujejo terminali s telekomunikacijskimi digitalnimi PCM potmi in preko katerega komunicirajo računalniki modulov medseboj pri obdelavi telekomunikacijskega prometa.

V pričujočem članku bomo pokazali rešitev medračunalniške komunikacije. Osnovna naloga sistema medprocesorske komunikacije je obdelava telekomunikacijskega prometa, to je vzpostavljanje in rušenje telekomunikacijskih zvez med terminali, ki so krmiljeni z različnimi procesorji. Sistem ISKRA 2000 realizira telekomunikacijski promet med terminali različnih modulov tako, da si procesorji modulov izmenjujejo sporočila, ko želijo ugotoviti pozvani terminal in telekomunikacijsko pot tako, da moduli pošiljajo sporočila samemu procesorju skupinskega stikala, ko želijo vzpostaviti telekomunikacijsko pot.

Poleg te osnovne naloge pa sistem medprocesorske komunikacije služi tudi za prenašanje
- sporočil o napakah, odkritih med sprotnim preskušanjem ali v času preskušanja na zahtevo vzdrževalnega osebja
- statističnih sporočil in sporočil o meritvah prometa ter
- sporočil za administriranje podatkovnih baz v modulih

Sistem medprocesorske komunikacije je sestavljen iz serijskih asinhronih popolnoma dupleksnih kanalov med procesorji modulov in skupinskim stikalom ter iz programov v računalniku skupinskega stikala in v računalnikih modulov.
Sistem medprocesorske komunikacije je del operacijskega sistema v sistemu ISKRA 2000.

Topologija medprocesorske komunikacijske mreže je zvezdasta: obstoji povezava od vsakega modula k skupinskemu stikalu preko serijskih asinhronih popolnoma dupleksnih kanalov. Zaradi zanesljivosti sistema je sistem medprocesorske komunikacije podvojen: imamo dva skupinska stikala in k vsakemu od njiju svoj komunikacijski kanal od vsakega modula. Konfiguracijo mreže glej na sliki 1.

Protokol medprocesorske komunikacije je z ozirom na semantiko sporočil popolnoma transparenten zaradi številnih zgoraj naštetih nalog medprocesorske komunikacije. Zato ocenjujemo, da je protokol primeren tudi za druge procesne sisteme, ki so realizirani z računalniškim distribuiranim krmiljenjem.

2. PROTOKOL MEDPROCESORSKE KOMUNIKACIJE

2.1 Tipi in identifikacija sporočil

V protokolu medprocesorske komunikacije ločimo naslednje tipe sporočil:

a/ enozlogovne krmilne signale
b/ dvozlogovne krmilne signale
c/ večzlogovna sporočila

Tipi sporočil in mesto zloga v sporočilu so določeni z najtežjimi tremi biti zloga in sicer:

111 ..... enozlogovni krmilni signal
110 ..... prvi zlog dvozlogovnega krmilnega signala
10x ..... prvi zlog večzlogovnega sporočila
0xx ..... naslednji zlogi večzlogovnega sporočila oz. drugi zlog dvozlogovnega krmilnega signala

(x = 0 ali 1)

2.2 Usmerjanje sporočil

Sporočila se usmerjajo s pomočjo:

a/ fizičnih naslovov komunikacijskih kanalov v skupinskem stikalu, iz katerih programi medprocesorske komunikacije v računalniku skupinskega stikala ugotovijo naslov modula - računalnika, od koder sporočilo prihaja

b/ naslova modula, kamor je sporočilo namenjeno; ta naslov se nahaja v glavi sporočila - v prvem zlogu več večzlogovnega sporočila

Krmilni signali se izmenjujejo samo med moduli in skupinskim stikalom in ne med moduli samimi, zato ta sporočila ne vsebujejo naslovov.

Skupinsko stikalo si na podlagi naslovov, ki jih ugotovi iz a/ in b/ ustvari tim. programski kanal - preslikavo izvornega modula "modul $_i$" v ponorni modul "modul $_j$" in obratno. Programski kanal je trojka

(smer, modul $_i$, modul $_j$),

kjer je

$$\text{smer} = \begin{cases} 0 & \text{sporočilo gre iz modula } _i \text{ v modul } _j \\ 1 & \text{sporočilo gre iz modula } _j \text{ v modul } _i \end{cases}$$

"modul $_i$" je indeks v tabeli programskih kanalov, s pomočjo katerega skupinsko stikalo ugotovi modul in smer sporočila. Na ta način je možno, da vsi naslednji zlogi sporočila ne vsebujejo naslov, kamor je zlog namenjen.

Ponorni modul ugotovi iz glave sporočila - prvega zloga sporočila, od katerega modula bodo prispeli naslednji zlogi. To glavo sporočila je izdelalo skupinsko stikalo, potem ko je ustvarilo programski kanal.

2.3 Format sporočila

Transparentnost podatkov v sporočilu dosežemo tako, da imamo poseben format sporočila, ki določa tip, začetek in konec informativnega dela sporočila - podatkovnih zlogov.

1. zlog : 1, 0, naslov izvornega / ponornega modula
2. zlog : 0, tip sporočila
3. zlog : 0, število podatkovnih zlogov k-5
4. zlog : 0, znakovni biti podatkovnih zlogov
5. zlog : 0, podatek brez znakovnega bita
.
.
.
k. zlog : vzdolžna paritetna kontrola

Ker v podatkovnih zlogih sporočila (glej 2.1) ne moremo prenašati tudi znakovnih bitov, jih zberemo v posebni zlog - znakovni zlog. Tako nam opisani format sporočila omogoča prenos do sedem podatkov. Če želimo prenesti več podatkov, jih moramo pakirati v več sporočil.

2.4 Potrjevanje zlogov in sporočil

Sprejem in pošiljanje zlogov med modulom in skupinskim stikalom morata biti sinhronizirana medseboj, kar pomeni, da se naslednji zlog ne sme odposlati, dokler ni sprejet prejšnji zlog na nasprotni strani. To sinhronizacijo izvršimo s tim. pozitivnim potrditvenim signalom PACK, ki pomeni oddajni strani:

- zlog je na sprejemni strani sprejet brez napak in
- odpošlji naslednji zlog

Če je sporočilo pravilno sprejeto, pošlje ponorni modul skupinskemu stikalu signal PEOM oziroma signal ASWR. Signal PEOM (pozitivni konec sporočila) je signal, ki pomeni, da je sporočilo pravilno sprejeto in skupinsko stikalo, ko sprejme ta signal, sprosti programski kanal. Signal ASWR pomeni, da mu bo sledil odgovor in ob sprejemu tega signala skupinsko stikalo obrne smer programskega kanala.

2.5 Kontrola napak

Kontrola napak odkriva napake, ki so nastale med prenosom zloga (paritetna napaka, premaknitev okna) oziroma logičnih napak (prekrivanje zlogov, neveljaven zlog z ozirom na stanje komunikacije in z ozirom na smer komuniciranja).

Če odkrije sprejemna stran napako med prenosom zloga, pošlje negativni potrditveni signal NACK, ki pove oddajni strani, naj ponovno pošlje odposlani zlog. Po treh zaporednih napakah med prenosom, se pošlje iz oddajne strani signal CLR v skupinsko stikalo, ki poruši programski kanal.

Med prenosom sporočila se računa vrednost vzdolžne paritete bitov v zlogih in se ob koncu sporočila primerja s sprejeto vrednostjo vzdolžne paritete. Če se ti dve vrednosti razlikujeta, je sporočilo nepravilno sprejeto in se odpošlje namesto PEOM signala signal o negativnem koncu sporočila NEOM.

Ko izvorni modul sprejme NEOM signal, prične sporočilo ponovno pošiljati. Če sprejme dvakrat zaporedoma NEOM signal, programski kanal poruši s CLR signalom.

Med pošiljanjem zlogov teče t.im. časovna kontrola do sprejema naslednjega zloga oz. do sprejema potrditvenega signala. Če se ta časovna kontrola izteče pri izvornem modulu, se pošlje signal CLR proti skupinskemu stikalu, ki ob sprejemu tega signala poruši programski kanal.

Tako moduli kot skupinsko stikalo lahko odkrijejo logične napake, tako da ob vsakem sprejetem zlogu ugotavljajo, če je zlog veljaven glede na tekoče stanje, smer komuniciranja ter na trenutno konfiguracijo sistema. V primeru logične napake, se programski kanal avtomatično poruši.

V vseh primerih napak (trikrat NACK, dvakrat NEOM, iztek časovnih kontrol, logične napake) poskuša izvorni modul vzpostaviti komunikacijo preko drugega skupinskega stikala. Oba skupinska stikala delujeta po principu "delitve obremenitve". Izvorni modul izbere pri vzpostavljanju komunikacijske poti vedno prvo prosto skupinsko stikalo.

2.6 Kontrola toka sporočil

Ker modul pošlje naslednji zlog sporočila šele, ko je sprejet PACK signal, smo se s tem izognili možnosti, da bi izvorni modul pošiljal zloge, ne da bi ponorni modul prejšnje zloge sprejel. Skupinsko stikalo vedno gleda, če je ponorni modul pravilno sprejel zlog, predno mu pošlje naslednji zlog oziroma predno pošlje PACK izvornemu modulu. Izvorni modul sprejme torej PACK šele, če je prejšnji zlog bil pravilno sprejet v ponornem modulu.

Ker po istem kanalu med modulom in skupinskim stikalom lahko tako sprejemamo kot oddajamo sporočilo in ko modul sprejme po odposlani glavi sporočila iz skupinskega stikala glavo sporočila namesto potrditveni signal, ima sprejem sporočila prednost pred oddajo sporočila. To pomeni, da se modul v tem primeru pripravi na sprejem sporočila, medtem ko odloži pošiljanje sporočila, ki ga je pričel oddajati. S tem smo preprečili kopičenje zahtev za komuniciranje v skupinskem stikalu.

2.7 Uvrščanje zahtev za komuniciranje

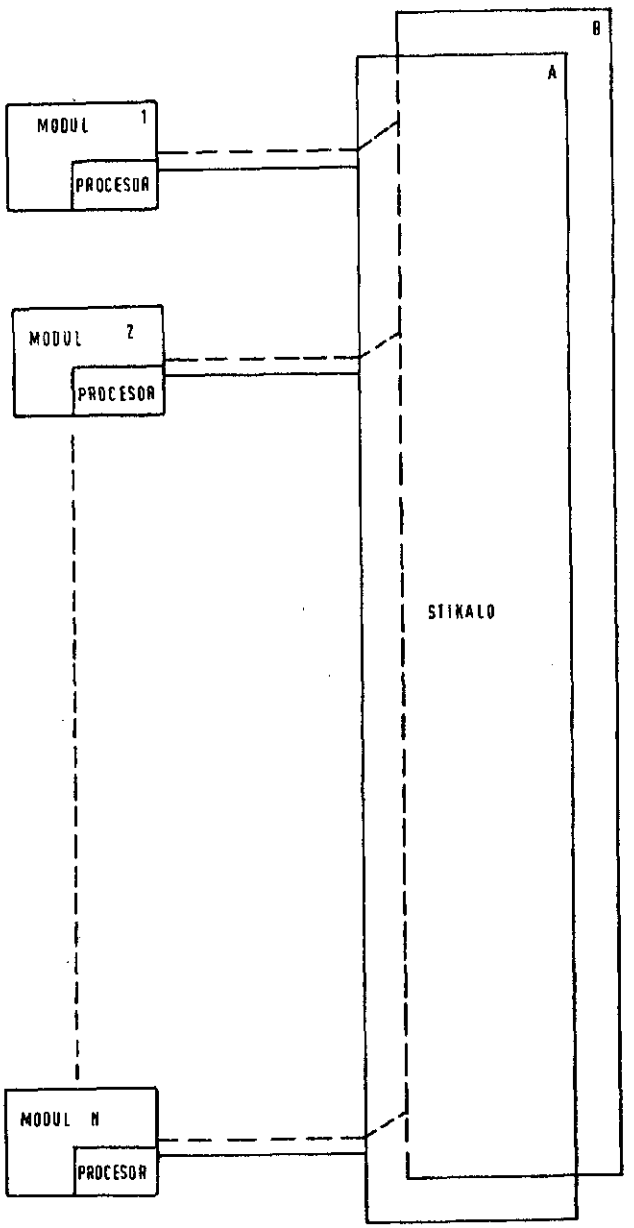
V modulu se generirajo sporočila iz večih psevdoparalelnih procesov, zato se sporočila v modulu uvrščajo v t.im. "izhodno čakajočo vrsto" od koder se jemljejo, ko modul sprejme po odposlanem sporočilu iz skupinskega stikala signal za nadaljevanje pošiljanja sporočil CNT. Če sprejme modul signal zaustavljanja pošiljanja sporočil HLT pomeni, da ne sme odpošiljati sporočilo, ker želi nek drug modul komunicirati z njim. CNT oz. HLT pošlje skupinsko stikalo v izvorni in ponorni modul, ko sprejme iz ponornega modula PEOM signal.

Zaradi distribuiranega krmiljenja in popolnoma decentralizirane podatkovne baze ter zaradi asinhronega in neodvisnega delovanja procesorjev v modulih vedno obstoji možnost, da želeni ponorni modul že komunicira z nekim drugim modulom. Tedaj skupinsko stikalo vrne izvornemu modulu po sprejeti glavi sporočila signal zasedenosti BUSY ter zapiše izvorni modul v čakajočo vrsto ponornega modula. Ko bo modul prišel na vrsto za komuniciranje z želenim modulom, mu bo skupinsko stikalo poslalo signal sproščanja RLSD (dvozlogovni krmilni signal) z naslovom modula, s katerim lahko komunicira. Če sprejme modul signal BUSY, lahko pošilja sporočila proti drugim prostim modulom.

3. ZAKLJUČEK

V referatu smo podali grobi oris protokola medprocesorske komunikacije, ki ga uporabljamo v sistemu ISKRA 2000. Izognili smo se podrobnostim, ki bi lahko zakrile osnovne značilnosti protokola, ki smo jih želeli prikazati.

Protokol je realiziran in omogoča zanesljive medmodularne povezave ter obnovo sistema iz nenormalnih situacij, ki lahko nastopijo med delovanjem neodvisnih procesov.



SLIKA 1
TOPOLOGIJA MEDPROCESORSKEGA KOMUNICIRANJA V SISTEMU ISKRA 2000

INFORMATICA 2/1981

TESTIRANJE ENOT MIKRORAČUNALNIKA V PROIZVODNJI

F. NOVAK,
A. DOBRIN,
B. ROPRET,
T. BLAZNIK

UDK: 681.3-181.4.001.4

INSTITUT „JOŽEF STEFAN",
ISKRA TOZD RAČUNALNIKI

Opisan je postopek testiranja enot mikroračunalnika ID 1680. Testiranje se izvaja na posebno prirejenem testnem mikroračunalniku ID 1680. Testirano enoto povežemo s testnim računalnikom preko vmesnika, ki ji posreduje testne sekvence in odčitava rezultate testiranja. Če osnovni "GO - NO GO" test odkrije napako, sledijo testi za njeno lokalizacijo. Podrobneje je opisano testiranje mikroprocesorske enote.

TESTING THE UNITS OF THE ID 1680 MICROCOMPUTER IN PRODUCTION. A procedure for testing the units of the ID 1680 microcomputer is described. The testing is implemented on a specially designed microcomputer ID 1680. The unit under the test is connected to the testing computer via testing interface which transmits test sequences and monitors test results. If the basic "GO - NO GO" test detects a fault, localization tests are resumed. Testing of the microprocessor unit is described precisely underneath.

UVOD

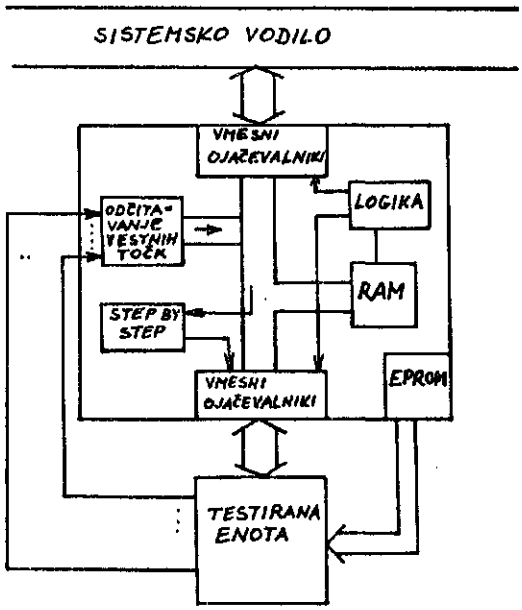
V proizvodnji mikroračunalnikov zavzema pomembno vlogo testiranje posameznih enot mikroračunalnika, kar je predpogoj za pravilno delovanje končnega produkta: ene od specifičnih konfiguracij mikroračunalnika.
Testiranje posamezne enote je v bistvu sestavljeno iz zaporedja testov, od katerih je prvi in obenem osnovni "GO - NO GO" test, ki detektira eventuelne napake v enoti. Ta test torej odloča, ali enota, ki je prišla iz proizvodnje, pravilno deluje, ali pa je na njej napaka. Zato je potrebno v njem podrobno preizkusiti vse funkcije, ki jih ima testirana enota. V primeru, da je bila odkrita napaka, sledijo testi za njeno lokalizacijo. Lokalizacija napak ni vedno uspešna in je možno, da pridemo v primeru večkratnih napak do napačnih zaključkov (protislovni rezultati testov). Takrat je potrebno enoto z napakami ročno pretestirati ob upoštevanju rezultatov avtomatskega testiranja.
Prvi test za lokalizacijo napake naj pokaže, če so v enoti fatalne napake, oziroma, če se enota sploh odziva na zunanje testne sekvence. Ostali testi preizkusijo delovanje posameznih funkcionalno zaključenih delov enote. Iz množice rezultatov sklepamo na lokacijo napak.

TESTIRANJE ENOT MIKRORAČUNALNIKA ID 1680

Testiranje se izvaja na posebno prirejenem testnem računalniku ID 1680. Enote, ki jih testiramo, povežemo s testnim računalnikom preko vmesnika, ki je na drugi strani priključen direktno na sistemsko vodilo (slika 1).

Na vmesniku za testiranje so realizirane naslednje funkcije:
- ojačitev sistemskih signalov in obenem ločitev testirane enote od sistemskega vodila testnega računalnika;
- logika za odjemanje podatkov iz testnih točk (signali, ki niso dostopni na konektorju);
- logika za določanje smeri pretoka signalov preko vmesnika (dvosmerne linije podatkovnega vodila, preklapljanje adresnih linij nad istim

pomnilnikom);
- 1 k bytov RAM pomnilnika za izmenjavanje testnih vzorcev in rezultatov testiranja med testnim računalnikom in testirano enoto;
- 4 k bytov EPROM pomnilnika za rezidenčne testne programe (z možnostjo razširitve);
- logika za nadziranje adresnega vodila EPROM pomnilnika (indikacija, če se testni program pravilno izvaja);
- vezje za izvajanje programa po korakih (step by step).



Slika 1

Enote, ki jih testiramo, ločimo na dva osnovna tipa: procesorske in periferne. Procesorske so tiste, ki vsebujejo mikroprocesor in so zmožne delovati samostojno, med periferne pa spadajo različne pomnilniške enote, vmesniki za periferijo in podobno.
Pri testiranju perifernih enot RAM in EPROM pomnilnika vmesnika ne uporabljamo. Logika vmesnika odpira vmesne ojačevalnike tako, da pridejo na konektor testirane enote kar signali sistemskega vodila (transparentno delovanje vmesnika). Testni računalnik dobiva rezultate testiranja direktno preko konektorja sistemskega vodila, ter dodatno informacijo iz testnih točk.

Podrobneje si oglejmo testiranje procesorskih enot:
Testirano enoto najprej preizkusimo z detekcijskim testnim programom (GO - NO GO) test. Ta program izvrši sama testirana enota, rezultat pa kontrolira testni računalnik, ki tudi sporoči, če je rezultat pravilen ali ne. Za izdelavo testnega programa je bila izdelana podrobna analiza funkcionalnega delovanja testirane enote. Med izvrševanjem testnega programa mora vsaka električna povezava med elementi enote vsaj enkrat zavzeti logično stanje "1" in stanje "0", dve sosednji povezavi naj izmenično zavzemata komplementarne vrednosti, rezultate posameznih akcij sproti kontrolira testirana enota sama, oziroma se odražajo v končnem rezultatu. Poleg analize testirane enote se ob izdelavi testnih programov upošteva tudi izkušnje in rezultate testiranja v tovarni, testni program je še dodatno usmerjen v odkrivanje najbolj pogostih napak.

Testni programi za lokalizacijo napak se izvršujejo korakoma pod kontrolo testnega računalnika. Podnožje EPROM pomnilnika testirane enote povežemo preko trakastega kabla z EPROM pomnilnikom vmesnika, kjer se nahajajo testni programi. Testirano enoto se resetira. Sledi izvajanje testnega programa po korakih. Tipična sekvenca testiranja je sledeča:
- testni rač. drži testirano enoto v stanju STOJ,
- testni rač. dovoli testirani enoti, da izvrši naslednjo instrukcijo testnega programa in zopet preide v stanje STOJ,
- testni rač. prečita informacijo o stanjih v testnih točkah ter o stanju adresnega vodila testirane enote in jo primerja z referenčnimi vrednostmi,
- sledi izvršitev naslednje instrukcije.

Izvajanje programa se prekine, ko se odčitane vrednosti ne ujemajo z referenčnimi. Iz mesta, kjer je bilo izvrševanje programa prekinjeno, sklepamo o lokaciji napake.

Izbira mesta testnih točk je rezultat analize posamezne enote, običajno jih izberemo na izhodih kompleksnejših funkcij (n. pr. izhod dekoderja adres in podobno).

LITERATURA

1) R.S.Bradley: A three stage spproach to LSI board testing , Electronic Engineering, April 1981
2) M.D.Lippmann, E.S.Donn: Design forethought promotes easier testing of microcomputer boards, Electronics, January 18, 1979

INFORMATICA 2/1981

JEDNA METODA PROCENE VREMENA TRAJANJA PRENOSA PODATAKA

VLADIMIR HOLODKOV

UDK: 681.3:621.398

ZOIL „VOJVODINA", RZ „VOJVODINA – AOP"
NOVI SAD

Članak daje jednu metodu procene vremena trajanja prenosa podataka telefonskim linijama. Razvijen je algoritam koji na svom ulazu kao promenljivu ima broj dokumenata čiji podaci su memorisani na nekom medijumu. Izlazna promenljiva je vreme potrebno za prenos memorisanih podataka.

A METHOD OF COMPUTING THE DATA TRANSMISSION TIME

The paper presents a method of calculation the data transmission time through phone lines. The algorythm is defined. A input variable of the algorythm is the number of documents which are stored on some of the storage media. A output variable is the data transmission time.

1 UVODNA RAZMATRANJA

U praksi se ukazala potreba da se pri projektovanju kapaciteta komponenti komunikacione mreže i njenoj eksploataciji, dobiju podaci o vremenu trajanja prenosa podataka i to telefonskim linijama. Kako je pri proceni postojala varijacija obima podataka, moralo se ići na generalizovanu formu procene vremena trajanja prenosa podataka. Algoritam je na svom ulazu imao obim podataka iskazan kao broj dokumenata čiji se memorisani podaci šalju a na izlazu se dobija vreme trajanja prenosa podataka. U tom posmatranju su brzina prenosa (u bitovi u sekundi) i dužina fizičkog sloga bili parametri. Za potrebw korektnog rada algoritma bilo je neophodno formirati "početno stanje" koje je izražavalo uticaje fizičkih karakteristika pojedinih komponenti komunikacione mreže.

Po preporukama CCITT koje važe i za SFRJ kao potpisnice, postoje standardme brzine prenosa podataka telefonskim linijama (baud rate) i definišu se svi uslovi potrebni za kvalitetan prenos podataka. Ali, vreme prenosa podataka se ne može dobiti jednostavnim deljenjem ukupnog broja bitova unetih dokumenata sa brzinom prenosa podataka koju diktira modem.

Uopšteni pristup rešavanju problema procene vremena trajanja prenosa je diktiran velikim varijetetima konfiguracije komunikacione mreže i dinamičkom promenom obima podataka koji se prenose.

Realna brzina prenosa podataka je ovisna od niza uticaja:

** Uticaj smetnji

U kontinulanom kanalu (fizička veza izmedju dva modema) se mogu posmatrati:

. linearna izobličenja
. nelinearna izobličenja
. smetnje:
  . aditivne
    -fluktuacioni šum (beli ili Gausov)
    -impulsni šumovi
    -interferentne smetnje
  . multiplikativne
    -varijacija prijemnog nivo signala (kratkotrajni prekidi)
    -frekvencijski pomeraj
    -fazni džiter
    -skokovi faze.

Svi ovi uticaji su u krajnjoj meri iskazani jednim parametrom koji se može nazvati "stopa pogrešnih bitova" (error bit rate) koji je ovisan od vrste telefonskog kanala i brzine prenosa podataka. Pored ovoga, CCITT daje indikaciju da pri prenosu podataka npr. preko iznajmljene telefonske linije dozvoljen broj ometajućih impulsa u jednom satu vršnog opterećenja iznosi oko 70. Kako je čest slučaj da se ometajući impulsi javljaju u paketima sa velikim pauzama izmedju njih, realno je posmatrati impulsne šumove kao da će se u toku jednog sata ponoviti slanje četiri bloka. Naime, CCITT kaže da pri pragu prenosa od -21dBm0 na iznajmljenoj liniji se može javiti 18 impulsa svakih 15 minuta, te ako se tih 18 impulsa posmatra kao jedan paket inkorporiran u blok podataka onda u jednom satu ima takvih 4 paketa.

** Uticaj upravljačkih znakova i sekvenci linijskog protokola (DLCC - Data Link Control Characters)

Na vreme trajanja prenosa utiče i broj upravljačkih znakova za uspostavljanje komunikacija izmedju dva terminalna uredjaja i sekvence iz konkretno primenjenog protokola. U zavisnosti od protokola (BSC, SDLC, HDLC itd.) menja se i broj upravljačkih znakova i sekvenci u jednom bloku podataka koji se prenosi u jednom trenutku. Ako je

linijski protokol primenjen na polu-dupleks tip prenosa podataka onda se javlja uticaj i obrtanja smera prenosa podataka (turnaround).

U cilju zaštite podataka od smetnji u jednom bloku podataka koji se prenosi, postoje razni postupci kontrole pariteta bitova na nivou znaka i na nivou bita (LRC, VRC, BCC ili CRC kontrola). To su redundantni znakovi te i oni utiču na vreme trajanja prenosa podataka.

** Uticaj hardverskih karakteristika uredjaja koji učestvuju u prenosu podataka

Modemi utiču na prenos podataka svojom karakteristikom "time turnaround" i "baud rate".

Periferni uredjaji (čitač kartica, jedinice magnetne trake, diskova, disketa i štampač) utiču na prenos podataka svojim karakteristikama u pogledu izvodjenja upisno/čitajućih (read/write) operacija. Ovaj uticaj je značajan ako nisu primenjene tehnike duple baferizacije ili preklapanja (overlaying) U/I operacija sa prenosom podataka.

Telefonska linija se može posmatrati kao talasovod te postoji vreme propagacije signala koje se povećava po odredjenoj zakonitosti sa porastom udaljenosti prijemnog terminala.

Vreme kašnjenja CPU može uticati i ono nije fiksno već promenljivo. Ovo zavisi od opterećenosti računara u toku trajanja prenosa podataka.

** Uticaj konverzije kodova

Prisutne su dve vrste konverzija u toku prenosa podataka:

- konverzija internog koda mašine u kod koji se koristi za prenos podataka (npr. ASCII u EBCDIC i obratno)
- konverzija paralelnog niza bitova koji formiraju jedan znak u serijski niz bitova koji ulaze u modem gde se vrši modulacija i obratno.

Uticaji ovih konverzija je zanemaren u primenjenom algoritmu jer se pošlo od pretpostavke da u većini današnjih uredjaja se koriste "bit slice" mikroprocesori.

** Uticaj manipulacije operatera sa memorijskim medijumima

Ukoliko se u toku prenosa podataka radi sa jedinicama magnetnih traka, čitačem kartica, štampačem ili drugim perifernim uredjajem, od interesa je posmatrati i vreme manipulacije operatera zbog potrebe npr. montiranja kotura magnetne trake i slično. Pretpostavljeno je da u toku prenosa podataka (linija je aktivna) u računskom centru je ostvarena takva organizacija rada da se na "host" računaru koriste jedinice magnetnih diskova a da se na udaljenim punktovima ne javlja potreba za manipulacijom sa memorijskim medijumima.

2 DOMEN PRIMENE ALGORITMA

Algoritam je bio prvobitno namenjen i praktično proveren pod sledećim okolnostima:

* nema "space compression"
* dvo-žični prenos preko komutirane telefonske linije
* polu-dupleks tip prenosa
* "point-to-point" komuniciranje
* BSC protokol koji emulira IBM 3780 terminal
* Remote Job Entry obrada
* nema overlayinga U/I operacija sa prenosom podataka.

Ali, u radu je data generalizacija koja se može primeniti i za druge uslove uz neznatne izmene algoritma.

3 OPIS ALGORITMA

U ovom radu su korišćene sledeće oznake:

- N_{dok} - broj dokumenata čiji memorisani podaci se prenose preko telefonske linije
- n_{REC} - jedan dokumenat se može posmatrati u smislu organizacije podataka na medijumima za pamćenje kao logički slog. Fizička organizacija podataka je obično drugačija tako da u jednom logičkom slogu ima n_{REC} fizičkih slogova
- L_{REC} - dužina fizičkog sloga na memorijskom medijumu u znakovima
- n_{DLCC} - maksimalan broj upravljačkih znakova primenjenog linijskog protokola u jednom bloku za transmisiju (bez sinhronizacionih znakova, redundantnih znakova za zaštitu - CRC i znakova za popunjavanje - pading)
- n_{SYN} - prosečan broj sinhronizacionih znakova u jednom bloku
- n_{PAD} - prosečan broj znakova za popunjavanje u jednom bloku
- n_{RED} - prosečan broj redundantnih znakova za proveru pariteta u jednom bloku
- n_{BIT} - broj bitova sa bitom pariteta po jednom znaku za kod koji se koristi za prenos podataka
- $n_{REC/BL}$ - maksimalan broj fizičkih slogova u jednom bloku
- t_{TA} - vreme obrtanja smera linije u sekundama (time turnaround)
- n_{SMER} - broj promena smera linije po jednom bloku koji se prenosi
- L_{blok} - maksimalna dužina bloka za prenos u znakovima
- t_{RD} - vreme čitanja jednog fizičkog šloga u sekundama
- t_{WRT} - vreme pisanja jednog fizičkog sloga u sekundama

Za vreme t_{RD} i t_{WRT} su potrebne napomene:

** jedinice diskova ili diskete (floppy disk)*

Srednje vreme pristupa upisno/čitajuće glave jednom cilindru je $t_{max}/3$ gde je t_{max} vreme kretanja u/č glave da stigne od prvog do poslednjeg cilindra. t_{max} se može i izračunati kao

$$t_{max} = n.t_o$$

gde je n broj cilindara za memorisanje unetih podataka sa dokumenata a t_o vreme potrebno u/č glavi da predje put izmedju dva susedna cilindra. Pored ovoga, javlja se uticaj rotacionog kašnjenja koje je dato sledećim izrazom:

$$60 / (2.N)$$

gde je N broj obrtanja osovine jedinice diska ili diskete u jednoj minuti. Kod jedinica disketa se javlja uticaj tzv. vremena postavljanja u/č glave (setting time) a to je vreme potrebno u/č glavi da predje iz neaktivnog u aktivno stanje (neaktivno stanje je odredjeno mehaničkom razdvojenošću u/č glave od površine diskete) i neka je to vreme t_p.

Prema tome, srednje vreme pristupa fizičkom slogu je dato aproksimativnim izrazom:

$$t_i = \frac{n.t_o}{3} + \frac{60}{2.N} + t_p \ (sec) \ (i=RD \ ili \ WRT)$$

* jedinice magnetnih traka

Prosečno vreme pristupa jednom slogu na magnetnoj traci je dato izrazom:

$$t_i = t_{pok} + \frac{N_z}{v.FB} + t_z \text{ (sec) (i=RD ili WRT)}$$

gde su:

- t_{pok} - vreme pokretanja kotura magnetne trake
- N_z - broj znakova u bloku upisanog ili učitanog sa magnetne trake
- v - brzina jedinice magnetne trake u znakovima u sekundi
- FB - faktor bloka (broj fizičkih slogova u jednom bloku)
- $N_z/(v.FB)$ - vreme potrebno da se svi znakovi fizičkog bloka prenesu sa magnetne trake u operativnu memoriju
- t_z - vreme zaustavljanja kotura magnetne trake

* čitač/bušač kartica

To je relativno spora jedinica te se radi sa tehnikom duplog baferisanja (u jednom baferu se formira blok podataka za prenos/prijem a drugom se nalazi blok koji se prenosi/prima). Broj bafera može biti veći od 2. Prosečno vreme formiranja jednog bloka za prenos može se dobiti aproksimativnim izrazom:

$$t_{FBL} = \frac{1}{x}\,\frac{60}{v}\,\frac{L_{blok}}{80} = t_i\,\frac{L_{blok}}{80} \text{ (sec) (i=RD ili WRT)}$$

gde su:

- x - broj bafera
- v - brzina čitanja/bušenja kartica u jednoj minuti

* štampač

To je takodje relativno spora jedinica namenjena isključivo za prijem podataka. U gornjem izrazu umesto 80 znakova treba da stoji npr. 132 ako se radi o štampaču sa 132 pozicije za štampanje a pri tome je v brzina štampanja redova u jednoj minuti.

Ostali podaci za potrebe rada algoritma su:

- t_{DELAY} - vreme kašnjenja CPU "host" računara je vreme koje centralni procesor upotrebi pre obrtanja linije a nakon prijema bloka podataka ("receive-to-transmit" kašnjenje). Ovo vreme može biti aproksimativno od nula do nekoliko sekundi, ovisno od brzine obrade podataka svih aplikacija na računaru, opterećenosti celog računara i od primenjenog tipa softvera za podršku komunikacijama
- t_{PROP} - vreme propagacije je vreme potrebno jednom slogu da se prenese izmedju modema. Ovo vreme je zavisno od medjusobnog rastojanja otpremnog i prijemnog terminala. Kreće se u granicama od 15 msec do 150 msec.
- t_{ACK} - vreme potrebno za prijem pozitivne potvrde primljenog bloka. Red veličine je 120 msec pri 600 bit/sec do 20 msec pri 2400 bit/sec
- Bd - standardna CCITT brzina prenosa na modemu u bitima u jednoj sekundi
- n_{GR} - inverzna stopa greške u bitima na telefonskoj liniji (koriste se CCITT preporuke). Na primer, na svakih $n_{GR}=10^3$ bita se može javiti jedan pogrešan bit pri brzini od 1200 bit/sec na komutiranoj telefonskoj liniji. To znači, da ako je dužina bloka za prenos 530 znakova i primenjen je EBCDIC kod tada se na svakoh prosečno jedan blok javlja jedan pogrešan blok koji se mora re-transmitovati.
- n_{IMP} - uticaj impulsnih smetnji će se iskazati kao broj ometajućih paketa impulsnih šumova na jedan sat. Broj ometajućih impulsa se može dobiti merenjima koja su složena i dugotrajna.
- T - vreme trajanja prenosa podataka za konkretan broj dokumenata uz sve prethodne predpostavke.

U razvijenom algoritmu (slika 1) je N_{dok} broj dokumenata posmatran kao nezavisno promenljiva a T kao zavisno promenljiva. Parametri su bili L_{REC} i Bd. Ostale promenljive su predstavljale konstante za odredjeni tip uredjaja, liniju i modeme.

Potrebna su sledeća izračunavanja:

- ukupan broj fizičkih slogova memorisanih na medijumu a koji će se slati preko telefonskih linija:

$$n_{FIZS} = N_{dok} \cdot n_{REC}$$

- ukupan broj znakova za prenos (bez redundantnih, DLCC, sinhronizacionih i popunjavajućih znakova):

$$n_{TZ} = (N_{dok} \cdot n_{REC}) \cdot L_{REC}$$

- ukupan broj blokova za prenos bez ponavljanja:

$$n_{TB} = \frac{n_{TZ}}{L_{blok}}$$

- broj blokova čije se slanje ponavlja zbog uticaja smetnji se može iskazati na sledeći način:

$$n'_{PB} = \frac{n_{TB}}{n_{GR}/(L_{blok} \cdot n_{BIT})} \text{ (uticaj smetnji)}$$

$$n''_{PB} = \frac{n_{IMP}}{3600} \cdot T \text{ (uticaj impulsnih smetnji)}$$

- ukupan broj blokova za prenos je tada:

$$N_{blok} = n_{TB} + n'_{PB} + n''_{PB}$$

- ukupan broj DLCC, sinhronizacionih, redundantnih i popunjavajućih znakova je:

$$n_{RZ} = (n_{DLCC} + n_{SYN} + n_{RED} + n_{PAD}) \cdot N_{blok}$$

- ukupan broj znakova koji se šalje u jednom smeru sa otpremnog terminala iskazanog u bitima je:

$$n_{TOTAL} = n_{BIT} \cdot (n_{TZ} + n_{RZ})$$

- ukupno vreme trajanja prenosa podataka bez korekcije je:

$$T' = \frac{n_{TOTAL}}{Bd} \text{ (sec)}$$

- ukupno vreme korekcije je: (slika 2)

$$T'' = (t_{RD} \cdot n_{REC/BL} + n_{SMER} \cdot t_{TA} + t_{WRT} \cdot n_{REC/BL} + t_{DELAY} + 2.t_{PROP} \cdot n_{REC/BL} + t_{ACK}) \cdot N_{blok}.$$

- ukupno vreme trajanja prenosa podataka je konačno:

$$T = T' + T''$$

Odredjenim zamenama i transformacijama se može dobiti zavisnost vremena trajanja prenosa podataka T od broja dokumenata N_{dok}, dok su parametri brzina prenosa podataka diktirana od strane modema Bd i dužina fizičkog sloga u znakovima L_{REC}.

Konačan izraz koji prikazuje vreme trajanja prenosa podataka je:

$$T = a.N_{dok}$$

gde su:

$$a = \frac{a_1 + a_2}{a_3}$$

$$a_1 = n_{REC} \cdot n_{BIT} \cdot \frac{L_{REC}}{Bd}$$

$$a_2 = \frac{L_{REC}}{Bd} \cdot \frac{n_{BIT} \cdot n_{REC}}{L_{blok}} \cdot \left(1 + \frac{L_{blok} \cdot n_{BIT}}{n_{GR}}\right)$$

$$a_3 = 1 - \frac{n_{BIT}}{Bd}\left(a_4 + a_5 \frac{Bd}{n_{BIT}}\right) \frac{n_{IMP}}{3600}$$

pri čemu su:

$$a_4 = n_{DLCC} + n_{SYN} + n_{RED} + n_{PAD}$$

$$a_5 = t_{RD} \cdot n_{REC/BL} + n_{SMER} \cdot t_{TA} + t_{WRT} \cdot n_{REC/BL} + t_{DELAY} + 2 \cdot t_{PROP} \cdot n_{REC/BL} + t_{ACK}$$

4 ESTIMACIJA ALGORITMA

Razvijen je program koji je omogućio dobijanje diskretnih vrednosti za vreme prenosa T i broj dokumenata N_{dok}. To su bile promenljive u programu. Dužina fizičkog sloga L_{REC} i brzina prenosa u modemu Bd su bili parametri. Posmatrana je konkretna komunikaciona mreža te je izvršen teorijski proračun po prikazanom algoritmu. Dobijene vrednosti vremena trajanja prenosa su uporedjivane sa vrednostima dobijene merenjem vremena na komutiranoj telefonskoj liniji izmedju dva grada na medjusobnom rastojanju od oko 100 km. Neki rezultati su prikazani kao dodatak ovom članku.

Merenja vremena trajanja prenosa podataka vršena su na komutiranom prenosnom putu koji se sastojao od dve čvorne telefonske centrale u Novom Sadu i Subotici, jedne krajnje telefonske centrale u Senti, dve VF parice, dve parice (jedna u Novom Sadu a druga u Senti) i jedne ARM veze u Novom Sadu od mesne telefonske centrale do čvorne (slika 3).

Za podatke dobijene u tabeli su važile sledeće vrednosti:
Korišćen je inteligentan terminal ADDS System 75 koji emulira BSC komuniciranje IBM 3780 i ima dve jedinice disketa kapaciteta oko 1 MBy. Vrednosti su: $n_{DLCC}=8$, $n_{SYN}=4$, $n_{BIT}=8$, $L_{blok}=530$, track-to-track vreme jedinice diskete je 3 msec, setting vreme je 15 msec, broj obrtaja osovine jedinice disketa je 360 obr/min, broj cilindara je 75. Upotrebljeni su modemi PP-1200 proizvodnje Instituta "Mihajlo Pupin" iz Beograda koji je imao $t_{TA}=15$ msec, broj obrtanja linije je $n_{SMER}=2$, a brzina prenosa je bila 1200 b/s. Ostale vrednosti su: $n_{REC}=1$, $L_{REC}=128$, $t_{ACK}=60$ msec, $t_{PROP}=20$ msec i $n_{GR}=10^4$. Broj ometajućih impulsa pri proračunu je zanemaren kao i vreme kašnjenja CPU t_{DELAY}.

Tabela 1: Rezultati merenja vremena na relaciji Novi Sad - Senta izvršeni 7.maja 1981.

Broj pokušaja uspostavljanja veze	Broj dokumenata	Procena vremena (sec)	Izmereno vreme (sec)	Razlika (sec)	% greške
2	112	164	165	1	0.6
6	132	193	225	32	16.5
4	366	536	570	34	6.3
isti poziv	550	805	810	5	0.6
"	690	1010	1035	25	2.4
"	1728	2530	2592	62	2.4
"	1701	2491	2721	230	9.2

Merenja vremena su vršena na još nekim relacijama i dobijene vrednosti su bile očekivane prema proračunu.

Analizom svih vremena trajanja prenosa podataka uzimajući u obzir kvalitet telefonskih linija u SAP Vojvodina došlo se do sledećih zaključaka:

* broj ometajućih impulsa na komutiranoj liniji u proračunu postaje značajan ako je vreme trajanja prenosa podataka veće od jednog sata. Taj uticaj je izražen i u prekidima na zahtev linijskog protokola jer su uslovi za prenos podataka pogoršani.

* Merenja su vršena uglavnom u prepodnevnim satima kada je telefonski saobraćaj intenzivan (period od 9,30 do 13,00 časova). Uticaj smetnji je postajao značajan u više od 30% utrošenog vremena na pokušaj slanja drugih datoteka. Zbog neuspeha se odustajalo od slanja velikih datoteka (više od 1700 dokumenata).

* Merenja su vršena na sledeći način: za svaku datoteku je biran telefonski pretplatnički broj na kraju linije gde se nalazio prijemni terminal. Time se želelo uspostavljanje raznih prenosnih puteva da bi se na neki način uprosečio uticaj smetnji. Odstupanja izmerenog vremena trajanja prenosa podataka od onog dobijenog primenom algoritma iz ovog rada je u proseku iznosilo oko +4% (uzimajući u obzir neobjavljena merenja). Ovakvo odstupanje se može smatrati zadovoljavajućim. Pozitivno odstupanje ukazuje na povećan broj pogrešnih bitova iskazanog sa veličinom n_{GR}.

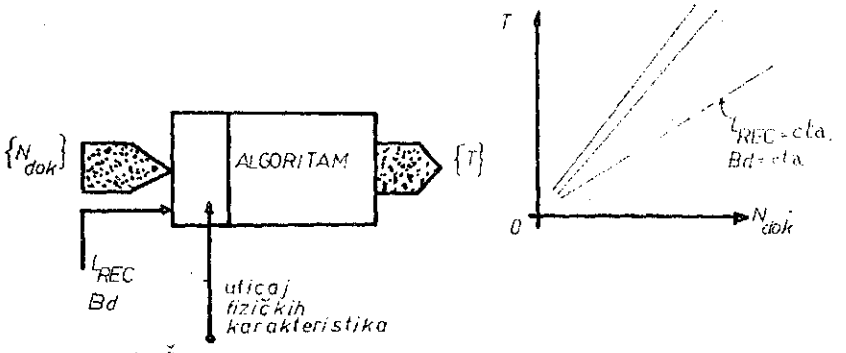
Osnovne vrednosti razvijenog algoritma su u sledećem:

* Obuhvaćen je niz faktora koji utiču na vreme prenosa podataka te se dobijaju tačnije vrednosti.

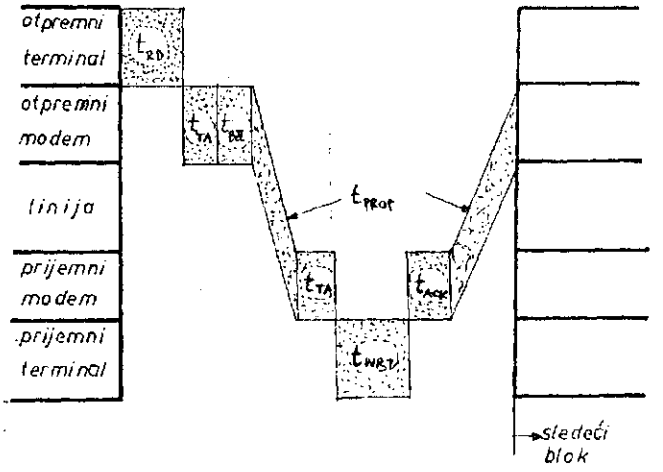
* Razvijen algoritam je praktično upotrebljen i rezultati su zadovoljavajući imajući na umu predpostavke navedene u sekciji 2.

* Algoritam je veoma jednostavan i programibilan i uzima u obzir dve veličine koje su od interesa za posmatranje sa aspekta korisnika komunikacione opreme: broj dokumenata i odgovarajuće vreme trajanja prenosa.

* Pri fiksnim vrednostima za brzinu prenosa podataka u modemu i dužini fizičkog sloga, moguće je formirati grafikon $T=f(N_{dok})$ tako da korisnici opreme su u mogućnosti da veoma brzo doznaju koliko vremena je potrebno da se prenese odredjen obim dokumenata. Ovo je naročito bitno ako se radi o velikoj radnoj organizaciji koja ima geografski dislocirane poslovne jedinice (npr. zajednice osiguranja, banke, elektrodistribucija i sl.) a koriste istorodnu opremu. Pored ovoga, poslovne jedinice su često i u raznim tarifnim zonama PTT saobraćaja te se na ovaj način olakšava i izračunavanje troškova prenosa podataka. Poslovne jedinice često imaju različite stepene prosečnog godišnjeg obima podataka tako da grafikon daje brzi odgovor na pitanje "Koliko vremena je potrebno da se prenese toliko i toliko dokumenata finansijskog knjigovodstva?".

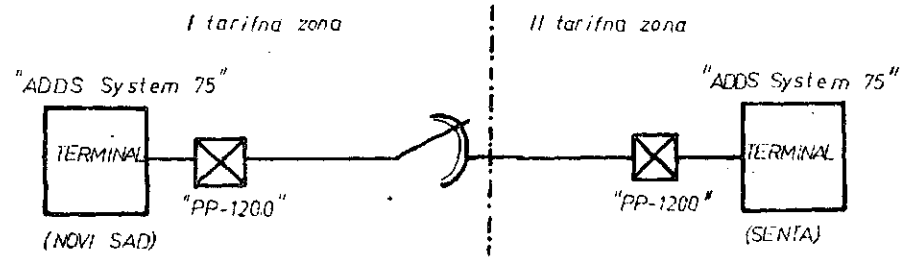


Slika 1 Šematski prikaz algoritam



Slika 2: Raspored vremena po jednom bloku

t_{Bd} : baud rate za jedan blok



Slika 3: Konfiguracija za vrednosti iz tab. 1

INFORMATICA 2/1981

SODOBNI DINAMIČNI POMNILNIKI ZA MIKRORAČUNALNIKE

R. MURN,
D. PEČEK

UDK: 681.3-181.4

INSTITUT „JOŽEF STEFAN", LJUBLJANA, JUGOSLAVIJA

V referatu je prikazana problematika zasnove in realizacije pomnilnega modula za sodobne mikroračunalnike. Osnovni gradniki modula so dinamični pomnilniki konfiguracije 64 K X 1. Referat je razdeljen v dve poglavji. Prvo predstavlja realizacijo vezij za odkrivanje in popravljanje enojnih napak v pomnilniku, ter odkrivanje dvojnih in masovnih napak. Drugo poglavje pa prikaže časovno pogojena krmilna vezja, ki skrbijo za optimalno reševanje konfliktnih situacij. Le-te so pogojene s poseganjem v pomnilnik (programsko ali DMA), pojavljanjem in popravljanjem napak, osveževanjem in besedno orientirano zaščito pomnilnika pred vpisom.

DYNAMIC MEMORY MODULES FOR UP-TO-DATE MICROCOMPUTERS. The design and realization problematics of dynamic memory module for up-to-date microcomputers are given in this paper. The first part deals with error detection and correction circuits for single, double and burst errors. In the second part the time domain circuits for arbitration in conflict situations are discussed.

UVOD

Dinamični pomnilniki (DRAM) so pomembni gradniki mikroračunalniških sistemov. Vedno večje razvojne zahteve narekujejo tudi vedno bolj kompleksne pomnilniške module. Dosedanji 4 K in 16 K pomnilniški čipi so uspešno v uporabi in predstavljajo industrijski standard. Nova generacije 64 K PČ nudi nadaljnje razvojne ugodnosti, prav tako pa zahteva nekatere specifične ukrepe v primeru pojavljanje trenutnih napak, katerih vzrok je v precejšnji meri odvisen od sevanja delcev alfa, saj tehnologija izdelave PC visoke integracije zahteva uporabo radioaktivnih elementov (radij, torij).

V referatu je opisan razvoj pomnilnega modula sestavljenega iz PČ velikosti 64 K x 1 z vsemi specifičnimi vezji potrebnimi za priključitev na gostiteljski mikroračunalnik. Glavni povdarek je na opisu vezja za korekcijo enojnih napak in detekcijo dvojnih in masovnih napak, ter reševanju časovnih pogojev, ki jih določa optimalno vsklajeno delovanje pomnilnega modula in hitre inačice mikroprocesorja 6800. Podane so rešitve kompleksne konfliktne situacije, ki jo narekujejo: poseganje v pomnilnik (programsko ali DMA), pojavljanje ter popravljanje napak in besedno orientirana zaščita pomnilnika pred vpisom.

Namen referata ni seznaniti bralca z uporabo in krmiljenjem dinamičnih PČ, pač pa prikazati kako bistveno povečamo zanesljivost delovanja pomnilnega modula, ga opremimo z dodatnimi funkcijami (zaščita, signalizacija) in kako z materialno opremo rešujemo konfliktne situacije v primeru, ko smo časovno zelo omejeni.

POVEČANJE ZANESLJIVOSTI DINAMIČNIH POMNILNIH MODULOV

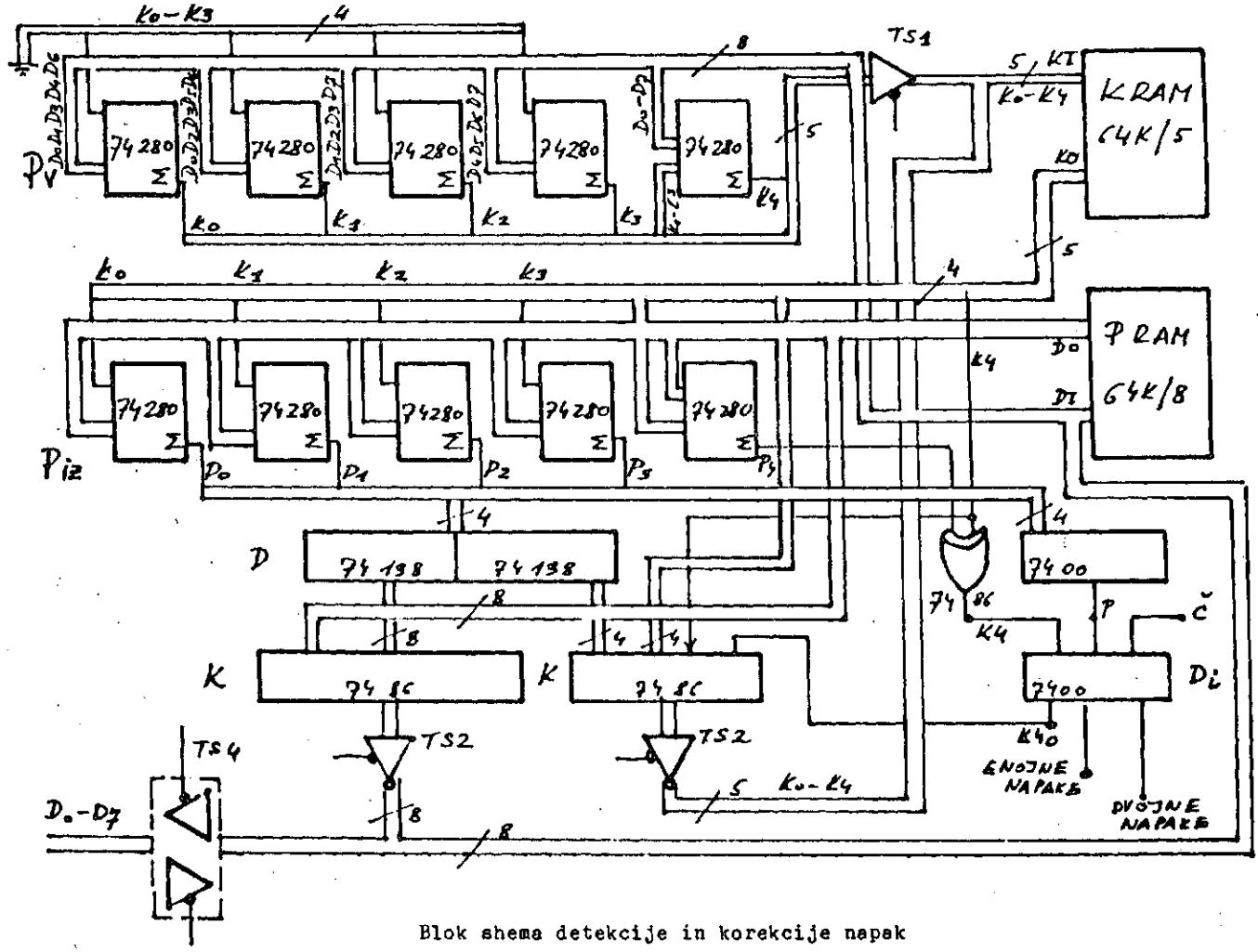
Sodobni 64 K PČ kar najbolj ustrezajo razvojnim zahtevam hitro razvijajočih se mikroračunalniških sistemov. Velika gostota pomnilnih celic, problem nečistoč, sevanje delcev alfa in kompleksnost vezij pa zahtevajo vpeljavo posebnih ukrepov za dosego visoke stopnje zanesljivosti. Izkušnje kažejo, da so najbolj pogoste enojne trenutne napake, ki so iz stališča diagnostike napak najbolj neugodne. Zastavi se nam vprašanje, kakšne testne postopke je potrebno izdelati,vendar tako, da ne povzročijo neprijetnih stranskih učinkov. Izkaže se, da so najbolj primerni detekcijski in korekcijski kodi. Enostavni in učinkoviti so Hammingovi kodi za korekcijo enojne in detekcijo dvojne napake. Osnovnim informacijskim bitom (m), dodamo preizkuševalne bite glede na parnost (k), tako, da dobimo potrebno število bitov n=m+k . Za dosego željenih lastnosti mora znašati Hammingova razdalja d=4 . S pomočjo izsledkov teorije detekcijskih in korekcijskih kodov ni težko določiti minimalno potrebno število bitov /1./. Za m=8 je k=5, kar predstavlja 62,5 odstotno redundanco. (Za samo korekcijo enojnih napak je k=4.)

Kako realizirati vezje, ki bo izpolnjevalo dane zahteve. Vezje mora biti dovolj enostavno in zanesljivo, upoštevati mora pojavitev napak na vseh bitih (m+k), čas potreben za korekcijo mora biti čim krajši.

V nadaljevanju je prikazana lastna zasnova sistema za korekcijo enojnih in detekcijo dvojnih napak. Uporabljen je 8 bitni mikro računalnik in 64 K x 1 PČ, torej 8+5=13 PČ. Sistem je zasnovan na originalni Hammingovi zamisli /2/, kjer se podatkovni in preizkuševalni biti med seboj prepletajo, tako, da je omogočeno enostavno dekodiranje napačnih bitov.

	Soda parnost	k_0	k_1	D_0	k_2	D_1	D_2	D_3	k_3	D_4	D_5	D_6	D_7	k_4
P_0		1	2	3	4	5	6	7	8	9	10	11	12	13
P_1	0	X		X		X		X		X		X		
P_2	0		X	X			X	X			X	X		
P_3	0				X	X	X	X					X	
P_4	0								X	X	X	X	X	
	0	X	X	X	X	X	X	X	X	X	X	X	X	X

Tabela 1



Blok shema detekcije in korekcije napak

Tvorimo naslednji vrstni red podatkovnih bitov (D_0 ÷ D_7) in preizkuševalnih bitov (k_0 ÷ k_4):

$$k_0 k_1 D_0 k_2 D_1 D_2 D_3 k_3 D_4 D_5 D_6 D_7 k_4$$

Kot je razvidno iz Tabele 1, imamo za korekcijo enojne napake 4 funkcije parnosti P_0÷P_3. Znak X pomeni za vsako funkcijo ustrezni podatkovni bit, ki se upošteva, ter ustrezni preizkuševalni bit, ki omogoči sodo parnostno funkcijo. Posamezni stolpci so binarno zakodirani in kažejo vrstni red bitov. Za detekcijo dvojnih napak je dodana funkcija parnosti P_4, ki jo določajo vsi podatkovni in preizkuševalni biti. Vezje za detekcijo je zgrajeno tako, da kot stranski efekt nudi detekcijo masovne napake "stuck at 1".

Pri zapisu v pomnilnik zapišemo podatke v PRAM in preizkuševalne bite, ki jih tvorimo s pomočjo vezij P_v v KRAM. Pri čitanju s pomočjo vezij P_{iz} primerjamo stara in novo tvorjene preizkuševalne bite. Če ni napak so vsi P_i enaki nič, prav tako je vrednost nič v točki K_4, ki zavzame vrednost nič tudi pri dvojnih napakah. S pomočjo signalov v točki P lahko ločimo oba primera. V primeru enojne napake v katerem koli bitu, zavzame ustrezen P_i vrednost 1. Mesto napake določi dekodirno vezje D, korekcijsko vezje K pa negira napačen bit. Napako v bitu k_4 lokaliziramo s pomočjo vhoda k_{40}. Vezje D_i omogoči diagnostiko enojnih in dvojnih napak. Dvojna napaka sproži alarm in prekinitev delovanja. Hitrost javljanja in korekcije napak je odvisna od hitrosti uporabljenih čipov. Tipična podatka za detekcijo in korekcijo sta 30-40 ns ter 50-60 ns.

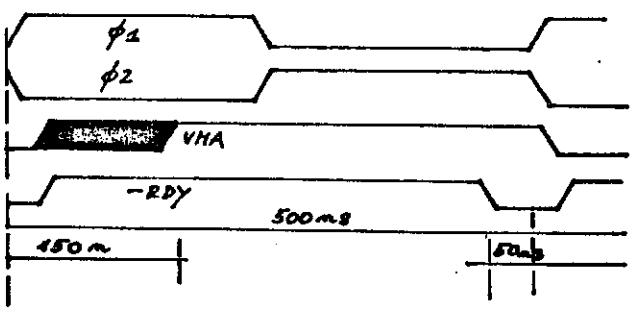
KRMILNA VEZJA DINAMIČNEGA POMNILNEGA MODULA

Slika krmilnih vezij za dinamične pomnilnike je v največji meri odvisna od tipa mikroprocesorja v sistemu in dodatnih zahtev in funkcij, ki naj jih pomnilni modul opravlja. V opisanem primeru gre za uporabo priljubljenega 8 bitnega mikroprocesorja 6800 in sicer njegove hitre verzije (6800H, 2MHz). Dodatna funkcija ki narekuje zgradbo krmilnih vezij je poleg detekcije in korekcije napak še besedno orientirana zaščita pred vpisom. Posebne zahteve pri gradnji modula pa so bile naslednje:

- kapaciteta modula 256K byt-ov
- delovanje mikroprocesorja ne sme biti upočasnjeno
- vsa krmilna vezja naj bodo izvedena s popularno serijo 74

Druga točka je narekovala še posebaj skrbno časovno analizo delovanja modula, posebaj zaradi velikoserijske proizvodnje, kjer ne moremo izbirati integriranih elementov kot pri izdelavi prototipa, niti nastavljati zakasnitev z monostabilnimi elementi.

Analiza je pokazala, da je s stališča projektiranje na "worst case" proizvajalca edina možna rešitev uporaba Shottky tehnologije za vsa časovno kritična vezja. Pri projektiranju je bilo seveda potrebno upoštevati tudi časovne zahteve krmiljenja PČ. Kot najbolj neugodna se je pokazala zahteva po minimalnem času mirovanja krmilnih signalov RAS in CAS (percharge time) med posameznimi aktivnimi fazami. Ta čas mirovanja nam precej zmanjša osnovno periodo 500 ns, ki nam je na voljo za izvršitev akcije v pomnilniku, saj je povsem normalno, da podsegamo v pomnilnik v dveh zaporednih ciklih. Vendar to še ni vse. Če nočemo upočasnjevati delovanja računalnika, moramo pravočasno zaključiti akcijo v pomnilniku s signalom "ready". Le-ta pa mora biti prisoten vsaj 30 - 50 ns prej, predno se cikel resnično zaključi. To pa pomeni, da se nam naših 500 ns zmanjša za dodatnih 50 ns. Različni proizvajalci PČ določajo različne čase mirovanja. Vendar so v primeru uporabe 6800H zanimivi samo tisti PČ, pri katerih je čas mirovanja manj kot 150ns. Razlog je preprost. Ob pričetku aktivne faze signala ϕ_1 še ne smemo posegati v pomnilnik, ker moramo počakati še na signal VMA, ki nam pove, da je adresa prisotna na vodilu. To specifikacijah proizvajalcev, pa lahko VMA kasni glede na ϕ_1 za največ 150 ns. To pa pomeni, da nam je ostalo na razpolago t_{ak}=500 - 150 - (30 do 50) t_{ak}= 300 do 320 ns. Razmere prikazuje slika 1.



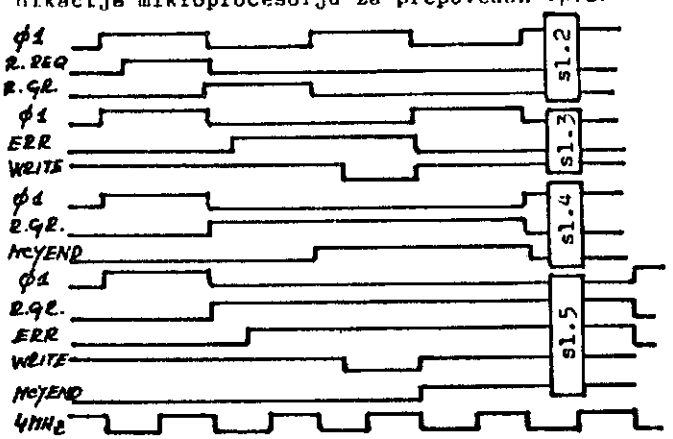
Slika 1

Seveda je t_{ak} tako kratek, da v tem času ne moremo razrešiti vseh konfliktov. Zato se je potrebno odločiti za smiselno arbitražo ob upoštevanju karakteristik pomnilnih čipov in mikroračunalnika. Le-te so naslednje:

- DMA lahko izvajamo le če je ϕ_1=1
- DMA lahko traja največ 4 us
- osveževanje med DMA ni dovoljeno
- vsi spominski cikli so tipa "read-modify"
  a) če je zahteva za "write" moramo najprej pogledati vrednost zaščitnega bita
  b) Če je zahteva za "read", pa moramo biti pripravljeni na popravljanje potencialne napake v pomnilniku.
- osveževalni cikel traja tako dolgo kot spominski cikel

Omenjene lastnosti nam vsilijo naslednje možne aktivnosti:

- med DMA prepovemo osveževanje (zaradi tega se zahteva po osveževanju javlja pogosteje) sl.2
- V primeru poseganja v pomnilnik brez zahteve po osveževanju in brez indikacije napake cikla ne podaljšamo (računalnik deluje s polno hitrostjo)
- če odkrijemo napako v pomnilniku, cikel podaljšamo za 250 ns da jo popravimo in pravilno besedo zapišemo nazaj v pomnilnik. (edino hitra verzija PČ firme Mostek je dovolj hitra, da bi napako lahko popravili v osnovnem ciklu - brez podaljševanja) sl. 3
- v primeru ko sovpadata poseg v pomnilnik in osveževanje cikel podaljšamo za 500 ns. sl. 4
- v primeru ko sovpadajo poseg v pomnilnik, napaka in osveževanje, cikel podaljšamo za 750ns sl. 5
- v primeru, ko odkrijemo dvojno ali masovno napako sledi signalizacija preko posebne linije
- v primeru zahteve za v pis v zaščiteno lokacijo cikel podaljšamo za 4.5 us, kar je indikacije mikroprocesorju za prepovedan vpis.



slike 2, 3, 4, 5

Sedaj se je potrebno odločiti še za vrstni red izvajanja operacij. Tu je potrebno upoštevati tip uporabljenih PČ. Če PČ omogoča t.i. skrito osveževanje (osveževanje tudi če je CAS aktiven) vrstni red aktivnosti ni pomemben. (To lepo lastnost imajo PČ tvrdke Mostek). Za vse ostale PČ pa velja, da ima najvišjo prioriteto osveževanje, šele nato pa lahko pričnemo s spominskim ciklom. Če iz kakršnih koli razlogov damo osveževanju najnižjo prioriteto, moramo podatke iz pomnilka ujeti v poseben register pred pričetkom osveževanja.

Važen podatek za dinamične pomnilnike je pogostost osveževanja. V našem primeru je izračun naslednji: 2ms/128 osveževanj = 15.6us/osvežitev. Ker v primeru DMA ne dopuščamo osvežitev more pogostost zahteve za osveževanje znašati 11.6us (15,6us - čas trajanja DMA(4us) = 11.6us). Dobljeno vrednost zaokrožimo na 11us.

Na osnovi prikazane analize lahko pristopimo k realizaciji posameznih vezij, ki v medsebojni povezavi predstavljajo krmilno funkcijo pomnilnega modula. Zaradi nekaterih časovno zelo kritičnih zahtev, je potrebno ustrezna vezja zasnovati tako, da delujejo kar najhitreje, čeprav

obseg uporabljenih elementov narasta (stremimo za paralelnim načinom delovanja). Paralelni način delovanja vodi v asinhronizem, zato je potrebno vpeljati dodatne informacijske signale, s pomočjo katerih delovanje krmilnika sinhroniziramo.

Poglejmo si podrobneje najvažnejša vezja, ki tvorijo krmilni del pomnilnega modula.

- detektor selekcije pomnilnega modula
- asinhroni generator signalov CAS, MUX
- generator selekcije posameznih bank
- vezje za nadzor osveževanja
- generator pisalnih impulzov in detektor konca spominskega cikla
- generator signala "ready"
- generator bita za zaščito
- generator asinhronih signalov DATA AVAIL, ERROR AVAIL, CORRECTED DATA AVAIL
- nadzornik podatkovnih pretokov
- generator notranjega osveževanja
- generator zunanjega osveževanja

Selekcija spominskega modula je hitro vezje, ki pogojuje selekcijo ene od štirih bank. Zaradi časovnih omejitev vezje aktiviramo že v delu periode, ko je ϕ_1 še aktiven (high). S pomočjo signala, ki pove, da je spominski modul selektiran in dveh dodatnih odrasnih linij A_{16} in A_{17} selektiramo eno od bank.

Po specifikacijah proizvajalcev določimo kasnitev med signalom RAS, ki ga generira prvo vezje in signaloma MUX in CAS. Pri načrtovanju kasnitve moramo upoštevati "worst case" kasnitve signala RAS, tako da čas dostopa glasi na RAS ne pa na CAS. Točna izvedba zakasnitve je možna samo z monoflopom ali pa z zakasnilnimi linijami. Serijsko proizvodnjo pa opravičujejo zakasnilne linije.

Nadzor osveževanja je izveden s števcem, ki ga poganja takt 4MHz. Ko števec prešteje do vrednosti, ki pogojuje trenutek za osveževanje se ustavi in generira signal REF REQ. Osveževanje omogočimo (REF GRANT), ko ϕ_1 preide iz 1 v 0.
Spominske in osveževalne cikle je potrebno strogo razmejiti. To razmejitev določa signal MCYEND (konec spominskega cikla). Sam spominski cikel lahko traja 500 ali pa 750 ns.

Kot smo že omenili, signal "ready" (RDY) pogoji konec aktivnosti v modulu. RDY lahko različno vpliva na dolžino trajanja aktivnosti v modulu. Možnih je osem kombinacij, saj je odvisen od treh signalov: napaka, osveževanje in zapis v zaščiteno lokacijo.

Spominski modul omogoča besedno orientirano zaščito pomnilnika. S stališča programske opreme sta to dve instrukciji, ki zaščito aktivirata ali pa deaktivirata. S stališča materialne opreme pa je zaščita enobitni register, katerega vrednost postane veljavna en cikel pred posegom v spomin in traja do konca spominskega cikla.

Zaradi asinhronih vezij je bilo potrebno vpeljati tri glavne krmilne linije, ki sinhronizirajo delovanje krmilnika. Linija DATA AVAIL pove, kdaj so podatki pri čitanju na internem vodilu, ERROR AVAIL pove kdaj lahko pogledamo v statusni register za indikacijo napak, CORRECTED DATA pa pove, kdaj je izzvenel prehodni pojav vezja za korekcijo.

Nadzornik podatkovnih pretokov je krmilno vezje, ki skrbi za pravilno preklapljanje ojačevalnikov treh stanj. Z njim krmilimo šest različnih ojačevalnikov, zato je potrebno konstrukciji tega vezja posvetiti precej pozornosti, predvsem zato, da zaradi hitrih preklopov ne prihaja do kratkih stikov na izhodih ojačevalnikov.
Spominski modul omogoča zunanje in notranje osveževanje. Zunanje osveževanje je izvedeno s sedem bitnim števcem, vezje za notranje osveževanje pa generira negativni signal RFSH. Za katero verzijo se odločimo, je odvisno od lastnosti uporabljenih PČ.

Ob koncu članka bi povdarili, da prikazani krmilnik deluje v večnivojski povratnih zankah in mora biti zaradi tega še posebaj skrbno načrtan in izdelan.

ZAKLJUČEK

Prikazana problematika gradnje sodobnega dinamičnega pomnilnika za mikroračunalnike posega v področje čiste materialne opreme. Članku nismo namenoma pridružili nobenih podrobnih načrtov, saj bi le-ti presegali okvir za katerega je članek namenjen. Bralec, ki se bo pri svojem delu srečal s podobno problematiko, bo morda v članku našel kak utrinek za nadaljnje delo.

LITERATURA

/1/ R. Murn; Postopki za povečevanje zanesljivosti digitalnih sistemov, Informatica 4, št. 2, 1980

/2/ R.W. Hamming; Error detecting and error correcting codes, Bell Syst. Techn. J., 29, jan. 1950

UPORABNI PROGRAMI

S to številko Informatice začenjamo objavljati redno rubriko za uporabne programe. Odgovorimo najprej na vprašanje, kaj je uporaben program in katera so uporabna področja računalniške uporabe.

Začnimo z uporabnimi področji! Med nje sodijo prav gotovo poslovna metodologija, ekonometrični izračuni, napovedovanja razvoja gospodarskih dogodkov in njihovih teženj, planiranje proizvodnje in postopkov, mali informacijski sistemi, metodologija obdelave podatkov, upravljanje podatkovnih zbirk in baz, uporaba v tehnologiji itd.

Konkretni uporabni programi so npr. programi za mali poslovni sistem, za glavno poslovno knjigo, za evidenco prihodkov in izdatkov, za mrežno planiranje, za obračunavanje osebnih dohodkov, za izračun različnih statističnih kazalcev, za urejanje podatkovnih zbirk, tudi krajši zabavni programi, ki kažejo določeno programirano tehniko, programi za preizkušanje računalniških virov in naprav, za kriptografiranje, za prikazovanje trendov na zaslonu terminala in tiskalniku, za urejanje zbirk itd. Zlasti so zaželeni programi s področja računalniške metodologije ter zanimivi programi s širokega področja tehnoloških aplikacij.

Rubrika "Uporabni programi", naj bi spodbujala avtorja, da napiše kratek prispevek v dokaj standardni obliki, in sicer:

- kratek opis področja, na katerem se program uporablja s pripadajočo metodologijo
- programska lista s komentarji (visok programirni ali zbirni jezik)
- izvajanje programa na dovolj ilustrativnem primeru, po možnosti v realnem okolju.

Programirni jeziki so lahko visoki in zbirni jeziki, npr. PL/I, ALGOL, COBOL, FORTRAN, PASCAL, FORTH, LISP itn. Na začetku programske liste naj bo v komentarju natanko označen tip programirnega jezika in njegova verzija oziroma podatki o prevajalniku ali zbirniku na določenem sistemu. Npr.:

PL/I-80, VERS. 1.3, CP/M 48k

Avtorji prihodnjih uporabnih programov se naprošajo, da upoštevajo predpisano obliko svojih prispevkov, kot bo razvidna iz objavljenih primerov. Vsak programski primer bo dobil tudi posebno označitev in sicer:

⟨designator⟩ ::=
Informatica UP ⟨tekoča številka⟩,
⟨ime programa⟩, ⟨datum⟩,
⟨ime avtorja⟩, ⟨opombe⟩

Designator bo prispevku dodalo uredništvo tik pod naslovom prispevka.

Sintaksa prispevka naj bi bila tale:

⟨naslov prispevka⟩
⟨designator⟩
⟨opis področja⟩
⟨programska lista⟩
⟨izvajanje programa⟩
⟨literatura⟩

VREDNOTENJE PROGRAMA IN METODA PREGLEDA (PERT)

Informatica UP 1
PERT
julij 1981
A.P.Železnikar
sistem CP/M, CBASIC2

1. Področje uporabe programa

Program PERT (UP1) izračuna najkrajši čas, ki je potreben za izvedbo določenega projekta pri danih pogojih ter določi verjetnost končanja projekta v času, ki ga vstavimo kot željeni čas končanja. Program izračuna časa poznega začetka, zgodnjega in poznega končanja za dejavnosti, kakor tudi čas mrtvila ter standardno deviacijo pričakovanih časov dejavnosti.

Pred uporabo programa si načrtamo projekt, tako da uporabimo grafično PERT metodo ali prednostno tabelo. Pred uporabo programa vstavimo število dejavnosti projekta vključno z nemimi dejavnostmi. Za vsako dejavnost vstavimo njeno začetno in končno vozlišče ter čase njenega trajanja, ki so ocenjeni kot optimistični, normalni in pesimistični čas.

Program lahko obravnava več sto dejavnosti, pri tem lahko modificiramo stavek

60 DIM A (I, 2), S (I), F (I), E (I, 2)

tako da izberemo I.

2. Primer načrta prototipne izdelave mikroračunalnika

Obravnavani primer obravnava prototipno izdelavo mikroračunalnika, ko je bil ta laboratorijsko že razvit, potrebno pa je izdelati demonstracijski prototip. Sestavimo si najprej tabelo dejavnosti (glej tabelo 1), ki kaže podatke vseh dejavnosti izdelovalnega postopka.

Na osnovi tabele 1 lahko narišemo usmerjeni graf, ki ga imamo na sliki 1. Ta graf ima posebej označene kritične dejavnosti, usmerjena povezava med dvema vozliščema pa vselej 4 podatke, in sicer: nad njo je vpisana številka dejavnosti iz tabele 1, pod njo pa pripadajoči trije časi. S tem smo tabelo 1 v celoti preslikali v graf na sliki 1.

3. Kratek opis programa in njegovo izvajanje

Program na listi 1 je napisan za prevajalnik (kompilator) CBASIC2 ter pojasnjuje sam sebe. Ta program ne uporablja zbirčne strukture in je izrazito interaktiven: s konzole se vstavljajo podatki in se dobivajo odgovori, kot kaže lista 2, ko se program izvaja. Ta program je tako uporabljiv za večino BASIC sistemov z minimalnimi spremembami. Višja oblika tega programa bi bila ta, da bi npr. oblikovali zbirke za tabelo dejavnosti (tabela 1), t.j. za vhodno tabelo in zbirko za izhodno tabelo s podatki iz liste 2. Te zbirki bi shranili na disketo tako, da ju je moč odčitavati iz zaslona. Seveda pa bi ju lahko tudi izpisali. Takšen program bomo pokazali v enem od kasnejših prispevkov.

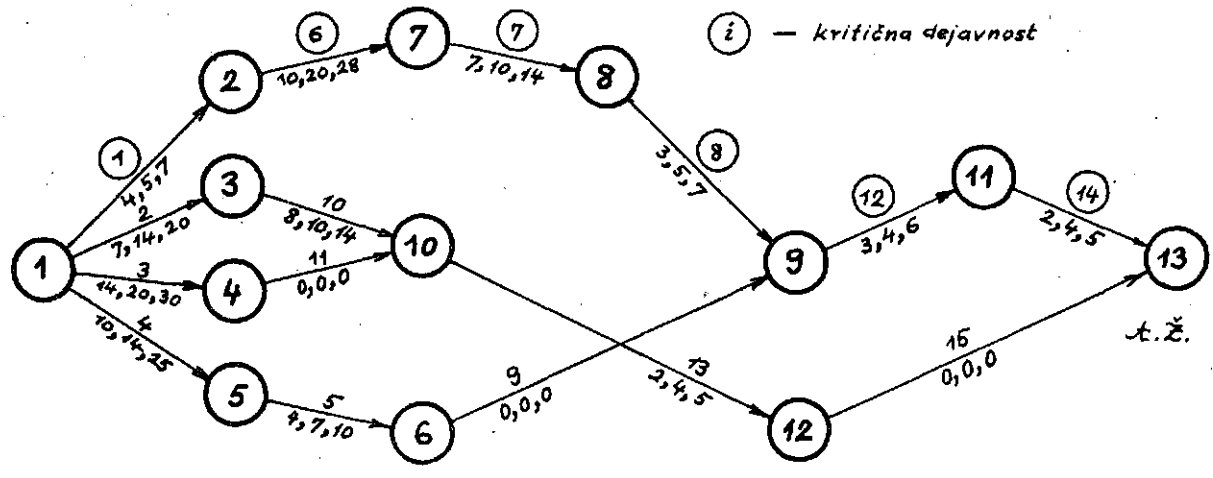
Program na listi 1 je modificiran program iz dela [1].

Literatura

[1] Program Evaluation and Review Technique (PERT), Practical Basic Programs (Ed. Lon Poole), Osborne/McGraw-Hill, 1980.

DEJ	VOZLIŠČI	ČAS (DNEVI)			POJASNILA DEJAVNOSTI
		OPT	NOR	PES	
1	1 - 2	4	5	7	Risanje logičnih/tehničnih načrtov
2	1 - 3	7	14	20	Dobava operacijskega sistema
3	1 - 4	14	20	30	Dobava uporabniške progr.opreme
4	1 - 5	10	14	25	Nabava materiala in diskov
5	5 - 6	4	7	10	Sestava ohišij in diskov
6	2 - 7	10	20	28	Izdelava tiskanih plošč
7	7 - 8	7	10	14	Dodelava plošč z elementi
8	8 - 9	3	5	7	Končno sestavljanje aparat. opr.
9	6 - 9	0	0	0	Nema dejavnost
10	3 - 10	8	10	14	Modifikacija operacijskega sistema
11	4 - 10	0	0	0	Nema dejavnost
12	9 - 11	3	4	6	Preizkušanje aparat. opreme
13	10 - 12	2	4	5	Preizkušanje program. opreme
14	11 - 13	2	4	5	Končanje mikrorač. sistema
15	12 - 13	0	0	0	Nema dejavnost

Tabela 1. Seznam dejavnosti in njihovih podatkov



Slika 1. Usmerjen graf za dejavnosti iz tabele 1, ki kaže kritično pot dejavnosti 1, 6, 7, 8, 12 in 14.

```
A_TYPE PERT.BAS
10 REM VREDNOTENJE PROGRAMA IN METODA PREGLEDA (PERT)
20 REM POLJE "A": ZACETNA IN KONCNA VOZLISCA DEJAVNOSTI
30 REM POLJE "S": CASI ZGODNJIH ZACETKOV DEJAVNOSTI
40 REM POLJE "F": CASI POZNIH KONCEV DEJAVNOSTI
50 REM POLJE "E": PRICAKOVANA TRAJANJA IN VARIANCE DEJAVNOSTI
60 DIM A(200,2),S(200),F(200),E(200,2)
65 DEF FNR(Z)=INT((Z!*1000+0.5))/1000
66 PRINT
70 PRINT "VREDNOTENJE PROGRAMA";
80 PRINT " IN METODA PREGLEDA (PERT)"
81 PRINT "=========================================="
82 PRINT "       (JULIJ 1981, ANTON P. ZELEZNIKAR)"
90 PRINT
100 PRINT "VSTAVI STEVILO DEJAVNOSTI";
110 PRINT " V DANI MREZI:";
120 INPUT N
130 PRINT
140 FOR I=1 TO N
150 PRINT
160 PRINT "----------------DEJAVNOST";I;"-----------------"
170 PRINT "VSTAVI ZACETNO IN KONCNO VOZLISCE:  ";
180 INPUT A(I,1),A(I,2)
190 IF A(I,2)<=A(I,1) THEN 220
200 IF A(I,2)<N THEN 280
220 PRINT "ZACETNO VOZLISCE MORA BITI NIZJE OSTE-"
230 PRINT " VILCENO KOT JE KONCNO VOZLISCE IN KON-"
240 PRINT " CNO VOZLISCE MORA BITI MANJSE OD STE-"
241 PRINT " VILA DEJAVNOSTI ..."
250 PRINT "********** POSKUSI PONOVNO **********"
260 PRINT
270 GOTO 150
280 PRINT "VSTAVI OBICAJNE TRI CASOVNE OCENE"
290 PRINT "  ZA DANO DEJAVNOST (A,M,B):  ";
300 INPUT A1,M,B
310 REM E(I,1) JE PRICAKOVANO TRAJANJE
320 E(I,1)=FNR((A1+M*4+B)/6)
330 REM E(I,2) JE VARIANCA DEJAVNOSTI
340 E(I,2)=FNR((B-A1)/6)^2
350 S(I)=0
360 F(I)=0
370 NEXT I
380 REM ZANKA ZA DOLOCITEV CASOV ZGODNJIH
381 REM  ZACETKOV V MREZI
390 FOR I=1 TO N
400 IF S(A(I,2))>=S(A(I,1))+E(I,1) THEN 420
410 S(A(I,2))=S(A(I,1))+E(I,1)
420 NEXT I
430 F(A(N,2))=S(A(N,2))
440 REM ZANKA ZA IZRACUN CASOV POZNIH
441 REM  KONCEV V MREZI
450 FOR I=N TO 1.0 STEP -1
460 IF F(A(I,1))=0 THEN 490
470 IF F(A(I,1))>F(A(I,2))-E(I,1) THEN 490
480 GOTO 500
490 F(A(I,1))=F(A(I,2))-E(I,1)
500 NEXT I
510 V=0
520 C=0
530 L=0
531 PRINT
540 FOR I=1 TO N
550 REM IZRACUN CASA MRTVILA V S1
560 S1=F(A(I,2))-S(A(I,1))-E(I,1)
570 PRINT "-------------------------------------------------------"
580 PRINT
590 PRINT "DEJAVNOST";I;"(OD VOZLISCA";A(I,1);
591 PRINT "DO VOZLISCA";A(I,2);")";" JE"
610 IF S1<=0 THEN 630
620 PRINT TAB(11);"NE";
630 PRINT TAB(13);"KRITICEN DOGODEK."
640 PRINT "PRICAKOVANO TRAJANJE: ";E(I,1);
650 PRINT TAB(31);"STAN. DEVIACIJA:";SQR(E(I,2))
660 IF S1>0 THEN 740
670 PRINT "!!!   NE ZACNI DEJAVNOSTI KASNEJE KOT PRI: ";S(A(I,1))
680 PRINT "!!!   DEJAVNOST SE MORA KONCATI PRI:  ";F(A(I,2))
690 REM ZBRANA DOLZINA POTI JE V L, VARIANCA V "V"
700 IF L>F(A(I,2)) THEN 720
710 L=F(A(I,2))
720 V=V+E(I,2)
730 GOTO 790
740 PRINT "ZGODEN ZACETEK:";S(A(I,1));
741 PRINT TAB(31);"POZEN ZACETEK:";
750 PRINT F(A(I,2))-E(I,1)
760 PRINT "ZGODEN KONEC:";S(A(I,1))+E(I,1);TAB(31);
770 PRINT "POZEN KONEC:";F(A(I,2))
780 PRINT "CAS MRTVILA (REZERVA): ";S1
790 NEXT I
800 PRINT
801 PRINT "**********************************************************"
810 PRINT "DOLZINA KRITICNE POTI JE ";L
820 P=SQR(V)
830 PRINT "PLUS ALI MINUS";P
840 PRINT "VSTAVI ZELENI DOVRSITVENI CAS (0 ZA IZSTOP)";
850 INPUT D
860 IF D<=0 THEN 1010
870 REM IZRACUNAJ Z-RAZMERJE ZA ZELENO TRAJANJE
880 Y=(D-L)/P
890 REM IZRACUNAJ ZBIRNO OBMOCJE PRI NORMALNI PORAZDELITVI
900 REM NAVEDBA: SOME COM.BASIC PROGR.,STR. 128
910 R=EXP(-(Y^2)/2)/2.5066282746
920 Z=Y
930 Y=1/(1+.33267*ABS(Y))
940 T=1-R*(.4361836*Y-(.1201676*Y^2)+.937298*Y^3)
950 IF Z>=0 THEN 970
960 T=1-T
970 PRINT "VERJETNOST DOVRSITVE S CASOM";
980 PRINT " TRAJANJA  === ";D;" ==="
991 PRINT "          ZNASA     ===== ";T;" ====="
990 PRINT
1000 GOTO 840
1010 END
```

Lista 1. Program PERT (točneje zbirka PERT.BAS) opravlja nalogo časovnega in verjetnostnega (mrežnega) planiranja delovnih (proizvodnih) postopkov, kot kaže lista 2 (ko se ta program izvaja). Program je napisan v jeziku BASIC (ta lista je v skladu z jezikom CBASIC2) za operacijski sistem CP/M (48k). Program pojasnjuje samega sebe in iz liste je razvidna njegova semantika.

```
A_CRUN2 PERT

CRUN VER 2.07P

VREDNOTENJE PROGRAMA IN METODA PREGLEDA (PERT)
**********************************************
      (JULIJ 1981, ANTON P. ZELEZNIKAR)

VSTAVI STEVILO DEJAVNOSTI V DANI MREZI:? 15


-----------------DEJAVNOST 1 ------------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 1,2
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 4,5,7

-----------------DEJAVNOST 2 ------------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 1,3
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 7,14,20

-----------------DEJAVNOST 3 ------------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 1,4
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 14,20,30

-----------------DEJAVNOST 4 ------------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 1,5
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 10,14,25

-----------------DEJAVNOST 5 ------------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 5,6
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 4,7,10

-----------------DEJAVNOST 6 ------------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 2,7
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 14,20,28

-----------------DEJAVNOST 7 ------------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 7,8
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 7,10,14

-----------------DEJAVNOST 8 ------------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 8,9
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 3,5,7

-----------------DEJAVNOST 9 ------------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 6,9
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 0,0,0

-----------------DEJAVNOST 10 -----------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 3,10
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 8,10,14

-----------------DEJAVNOST 11 -----------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 4,10
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 0,0,0

-----------------DEJAVNOST 12 -----------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 9,11
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 3,4,6

-----------------DEJAVNOST 13 -----------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 10,12
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 2,4,5

-----------------DEJAVNOST 14 -----------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 11,13
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 2,4,5

-----------------DEJAVNOST 15 -----------------
VSTAVI ZACETNO IN KONCNO VOZLISCE:  ? 12,13
VSTAVI OBICAJNE TRI CASOVNE OCENE
  ZA DANO DEJAVNOST (A,M,B):  ? 0,0,0

-------------------------------------------------------

DEJAVNOST 1 (OD VOZLISCA 1 DO VOZLISCA 2 ) JE
          KRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  5.167  STAN. DEVIACIJA: 0.5
!!!   NE ZACNI DEJAVNOSTI KASNEJE KOT PRI:   0
!!!   DEJAVNOST SE MORA KONCATI PRI:   5.167
-------------------------------------------------------

DEJAVNOST 2 (OD VOZLISCA 1 DO VOZLISCA 3 ) JE
        NEKRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  13.833 STAN. DEVIACIJA: 2.167
ZGODEN ZACETEK: 0             POZEN ZACETEK: 20.668
ZGODEN KONEC: 13.833          POZEN KONEC: 34.501
CAS MRTVILA (REZERVA):  20.668
-------------------------------------------------------

DEJAVNOST 3 (OD VOZLISCA 1 DO VOZLISCA 4 ) JE
        NEKRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  20.667 STAN. DEVIACIJA: 2.667
ZGODEN ZACETEK: 0             POZEN ZACETEK: 24.167
ZGODEN KONEC: 20.667          POZEN KONEC: 44.834
CAS MRTVILA (REZERVA):  24.167
-------------------------------------------------------

DEJAVNOST 4 (OD VOZLISCA 1 DO VOZLISCA 5 ) JE
        NEKRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  15.167 STAN. DEVIACIJA: 2.5
ZGODEN ZACETEK: 0             POZEN ZACETEK: 18.5
ZGODEN KONEC: 15.167          POZEN KONEC: 33.667
CAS MRTVILA (REZERVA):  18.5
-------------------------------------------------------
```

Lista 2. Ta lista in lista na naslednji strani predstavlja izvajanje programa PERT iz liste 1. Najprej se napiše naslov programa in vstavi število dejavnosti (iz tabele 1), nato pa se začno pisati zahteve posameznih dejavnosti za vstavljanje podatkov, in sicer za začetno in končno vozlišče vsake dejavnosti ter za tri pripadajoče čase (npr. dneve, ure, mesece itn.), ki se ocenjujejo kot optimistični, normalni in pesimistični. Ko je vstavljanje končano (s podatki dejavnosti 15, ki je zadnja), se začnejo izpisovati rezultati v obliki napotkov in podatkov, ko so izračuna za vsako dejavnost njeno pričakovano trajanje, standardna deviacija, njen začetek, konec in rezervni čas. Na koncu rezultatnega izračuna se izpiše še dolžina kritične poti in njen tolerančni interval (glej nadaljevanje liste na naslednji strani).

```
DEJAVNOST 5 (OD VOZLISCA 5 DO VOZLISCA 6 ) JE
        NEKRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  7        STAN. DEVIACIJA: 1
ZGODEN ZACETEK: 15.167          POZEN ZACETEK: 33.667
ZGODEN KONEC: 22.167            POZEN KONEC: 40.667
CAS MRTVILA (REZERVA):  18.5
-----------------------------------------------------

DEJAVNOST 6 (OD VOZLISCA 2 DO VOZLISCA 7 ) JE
          KRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  20.333 STAN. DEVIACIJA: 2.333
!!!   NE ZACNI DEJAVNOSTI KASNEJE KOT PRI:  5.167
!!!   DEJAVNOST SE MORA KONCATI PRI:  25.5
-----------------------------------------------------

DEJAVNOST 7 (OD VOZLISCA 7 DO VOZLISCA 8 ) JE
          KRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  10.167 STAN. DEVIACIJA: 1.167
!!!   NE ZACNI DEJAVNOSTI KASNEJE KOT PRI:  25.5
!!!   DEJAVNOST SE MORA KONCATI PRI:  35.667
-----------------------------------------------------

DEJAVNOST 8 (OD VOZLISCA 8 DO VOZLISCA 9 ) JE
          KRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  5        STAN. DEVIACIJA: 0.667
!!!   NE ZACNI DEJAVNOSTI KASNEJE KOT PRI:  35.667
!!!   DEJAVNOST SE MORA KONCATI PRI:  40.667
-----------------------------------------------------

DEJAVNOST 9 (OD VOZLISCA 6 DO VOZLISCA 9 ) JE
        NEKRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  0        STAN. DEVIACIJA: 0
ZGODEN ZACETEK: 22.167          POZEN ZACETEK: 40.667
ZGODEN KONEC: 22.167            POZEN KONEC: 40.667
CAS MRTVILA (REZERVA):  18.5
-----------------------------------------------------

DEJAVNOST 10 (OD VOZLISCA 3 DO VOZLISCA 10 ) JE
        NEKRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  10.333 STAN. DEVIACIJA: 1
ZGODEN ZACETEK: 13.833          POZEN ZACETEK: 34.501
ZGODEN KONEC: 24.166            POZEN KONEC: 44.834
CAS MRTVILA (REZERVA):  20.668
-----------------------------------------------------

DEJAVNOST 11 (OD VOZLISCA 4 DO VOZLISCA 10 ) JE
        NEKRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  0        STAN. DEVIACIJA: 0
ZGODEN ZACETEK: 20.667          POZEN ZACETEK: 44.834
ZGODEN KONEC: 20.667            POZEN KONEC: 44.834
CAS MRTVILA (REZERVA):  24.167
-----------------------------------------------------

DEJAVNOST 12 (OD VOZLISCA 9 DO VOZLISCA 11 ) JE
          KRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  4.167  STAN. DEVIACIJA: 0.5
!!!   NE ZACNI DEJAVNOSTI KASNEJE KOT PRI:  40.667
!!!   DEJAVNOST SE MORA KONCATI PRI:  44.834
-----------------------------------------------------

DEJAVNOST 13 (OD VOZLISCA 10 DO VOZLISCA 12 ) JE
        NEKRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  3.833  STAN. DEVIACIJA: 0.5
ZGODEN ZACETEK: 24.166          POZEN ZACETEK: 44.834
ZGODEN KONEC: 27.999            POZEN KONEC: 48.667
CAS MRTVILA (REZERVA):  20.668
-----------------------------------------------------

DEJAVNOST 14 (OD VOZLISCA 11 DO VOZLISCA 13 ) JE
          KRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  3.833  STAN. DEVIACIJA: 0.5
!!!   NE ZACNI DEJAVNOSTI KASNEJE KOT PRI:  44.834
!!!   DEJAVNOST SE MORA KONCATI PRI:  48.667
-----------------------------------------------------

DEJAVNOST 15 (OD VOZLISCA 12 DO VOZLISCA 13 ) JE
        NEKRITICEN DOGODEK.
PRICAKOVANO TRAJANJE:  0        STAN. DEVIACIJA: 0
ZGODEN ZACETEK: 27.999          POZEN ZACETEK: 48.667
ZGODEN KONEC: 27.999            POZEN KONEC: 48.667
CAS MRTVILA (REZERVA):  20.668

*****************************************************
DOLZINA KRITICNE POTI JE  48.667
PLUS ALI MINUS 2.82836825749
VSTAVI ZELENI DOVRSITVENI CAS (0 ZA IZSTOP)? 50
VERJETNOST DOVRSITVE S CASOM TRAJANJA  ===  50  ===
        ZNASA    =====  0.681274513491  =====

VSTAVI ZELENI DOVRSITVENI CAS (0 ZA IZSTOP)? 60
VERJETNOST DOVRSITVE S CASOM TRAJANJA  ===  60  ===
        ZNASA    =====  0.999968925415  =====

VSTAVI ZELENI DOVRSITVENI CAS (0 ZA IZSTOP)? 55
VERJETNOST DOVRSITVE S CASOM TRAJANJA  ===  55  ===
        ZNASA    =====  0.987414391335  =====

VSTAVI ZELENI DOVRSITVENI CAS (0 ZA IZSTOP)? 45

WARNING NE
VERJETNOST DOVRSITVE S CASOM TRAJANJA  ===  45  ===
        ZNASA    =====  0.0973905817168  =====

VSTAVI ZELENI DOVRSITVENI CAS (0 ZA IZSTOP)? 0
```

Lista 2 (nadaljevanje s prejšnje strani). Ko je izračun končan, lahko vtavljamo še podatke o želenih dovršitvenih časih (poteh), ki se razlikujejo od izračunanega časa kritične poti (pri verjetnosti dovršitve 0,5). V našem primeru, ko smo imeli kritično pot 48,667 plus/minus 2,828 , imamo pri vrednosti poti 50 verjetnost dovršitve /.68 , pri vrednosti poti 60 pa že verjetnost 0,99997. Pri podkritični dolžini poti 45 se najprej pojavi opozorilo, verjetnost dovršitve pa se zmanjša na vrednost 0.097. Izvajanje programa se konča, ko vtipkamo za dovršitveni čas vrednost 0. Dani program PERT lahko še v marsičem dopolnimo, tudi tako, da nam shranjuje podatke v obliki tabel v posebnih zbirkah na disku.

IZPISOVANJE SPLOŠNIH POLOŽNIC

Informatica UP 2
POLO
avgust 1981
B. Blatnik
sistem DELTA-M V1.2, BASIC2

1. Področje uporabe programa

Program izpolnjuje dvojne splošne položnice (obr. št. 10). Podatke črpa iz sekvenčne datoteke. Posamezni zapisi te datoteke imajo sledečo obliko:

priimek ime | ustanova | ulica št. | kraj | znesek ||

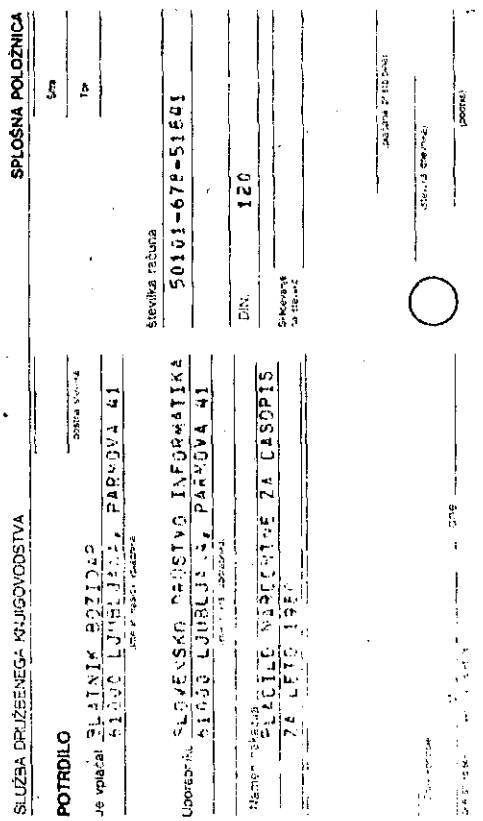
Če v kakšnem polju ni ustreznega podatka, je v tem polju znak "*".

2. Programska lista in kratek opis programa

Program POLO je sestavljen iz glavnega (POLO3)in dveh pomožnih modulov (POLO1 in POLO2).

Stavka 32 in 34 definirata vhodno oz. izhodno datoteko. Stavki 36 - 43 definirajo izpise, ki so enaki na vseh položnicah (konstantni nizi znakov). Spremenljive nize izvlečemo iz zapisov s pomočjo podprograma EXTR (modul POLO1). S podprogramom STORE (modul POLO2) postopoma tvorimo izhodno datoteko in lahko tudi izpisujemo položnice. Izhodno datoteko izpišemo s hitrim tiskalnikom s sistemskim programom PRINT.

```
1       SUB STORE(GAMA$)
2       REM POLO2
22      PRINT GAMA$
23      MOVE TO #2%,GAMA$=120%
24      PUT #2%
2000    SUBEND
```




```
1       REM POLO3 - MAIN
2       REM
10      DIM #1%,ALFA$(230%),BETA$(230%)
11      DIM #2%,GAMA$(120%)
30      ON ERROR GOTO 1000
31      CZ%=0.
32      OPEN "CASOPI.AUX" FOR INPUT AS FILE #
        #1%,ORGANIZATION SEQUENTIAL VARIABLE,#
        ACCESS READ,RECORDSIZE 230%
34      OPEN "PRINT.POL" FOR OUTPUT AS FILE #
        #2%,ORGANIZATION SEQUENTIAL VARIABLE,#
        ACCESS WRITE,RECORDSIZE 120%
36      STRAC$="50101-678-51841"
38      UPO1$="SLOVENSKO DRUSTVO INFORMATIKA"
40      UPO2$="61000 LJUBLJANA, PARMOVA 41"
42      NAMEN$="PLACILO NAROCNINE ZA CASOPIS"
43      CAS$="ZA LETO 1980"
44      GET #1%
46      MOVE FROM #1%,ALFA$=230%
48      CALL EXTR(ALFA$,ATME$,ANASLOV$,AZNESEK$)
49      CZ%=1
50      GET #1%
51      CZ%=0
52      MOVE FROM #1%,BETA$=230%
54      CALL EXTR(BETA$,BTME$,BNASLOV$,BZNESEK$)
56      AZ%=66-LEN(ATME$)
58      GAMA$=ATME$+SPACE$(AZ%)+BTME$
60      CALL STORE(GAMA$)
66      AZ%=66-LEN(ANASLOV$)
68      GAMA$=ANASLOV$+SPACE$(AZ%)+BNASLOV$
70      CALL STORE(GAMA$)
76      ARC$=SPACE$(120%)
78      PRINT\PRINT
80      MOVE TO #2%,ARC$=120%
82      PUT #2% \ PUT #2%
84      GAMA$=UPO1$+SPACE$(5%)+STRAC$+%
        SPACE$(17%)+UPO1$+SPACE$(5%)+STRAC$
86      CALL STORE(GAMA$)
92      GAMA$=UPO2$+SPACE$(39%)+UPO2$
94      CALL STORE(GAMA$)
96      PRINT
98      MOVE TO #2%, ARC$=120%
100     PUT #2%
102     GAMA$=SPACE$(40%)+AZNESEK$+SPACE$(63%)%
        +BZNESEK$
104     CALL STORE(GAMA$)
106     GAMA$=NAMEN$+SPACE$(38%)+NAMEN$
108     CALL STORE(GAMA$)
109     GAMA$=CAS$+SPACE$(54%)+CAS$
110     CALL STORE(GAMA$)
111     FOR K=1 TO 14
112     PRINT
114     MOVE TO #2%, ARC$=120%
116     PUT #2%
118     NEXT K
119     IF CZ%=1 THEN GOTO 1100
120     GOTO 44
1000    IF CZ%=0 THEN GOTO 1100
1001    BTME$=" "
1002    BNASLOV$=" "
1003    BZNESEK$=" "
1004    GOTO 56
1100    PRINT ERR
1102    PRINT ERL
1104    PRINT ERT$(ERR%)
2000    END
```

```
1       SUB EXTR (ZAPIS$,TME$,NASLOV$,ZNESEK$)
2       REM POLO1
4       Y%=POS(ZAPIS$,"*",1%)
5       IF Y%=1 THEN GOTO 10
6       X%=POS(ZAPIS$,"!",1%)
7       TME$=SEG$(ZAPIS$,1%,X%-1)
8       Z%=POS(ZAPIS$,"!",X%+1)
9       GOTO 12
10      Z%=POS(ZAPIS$,"!",3%)
11      TME$=SEG$(ZAPIS$,3%,Z%-1)
12      X%=POS(ZAPIS$,"!",Z%+1)
13      ULICA$=SEG$(ZAPIS$,Z%+1,X%-1)
14      Z%=POS(ZAPIS$,"!",X%+1)
15      KRAJ$=SEG$(ZAPIS$,X%+1,Z%-1)
16      X%=POS(ZAPIS$,"!",Z%+1)
17      ZNESEK$=SEG$(ZAPIS$,Z%+1,X%-1)
18      NASLOV$=KRAJ$+", "+ULICA$
2000    SUBEND
```

NOVICE IN ZANIMIVOSTI

Računalniška industrija v ZDA (1980)

V številki 2/1980 časopisa Informatica smo prikazali stanje računalniške industrije v ZDA za leto 1979. Letos ponavljamo to poročilo za leto 1980, izčrpne podatke za leto 1979 pa najdemo v Informatici 2/1980.

Leto 1980 označujejo v ZDA kot težavno leto za računalniško industrijo, in sicer zaradi gospodarske recesije v Ameriki in Evropi. Sto ameriških računalniških podjetij je proizvedlo 55,6 milijard dolarjev, kar je 20,4 % več kot v preteklem letu, pri tem se je povečal tudi obseg razvoja in raziskav (RR) za 22 %.

Tabela 1

10 največjih (v milijonih dolarjev)

		dohodek 1980	dohodek 1979	rast %
1	IBM	21.367	18.338	16,5
2	NCR	2.840	2.528	12,3
3	CDC	2.791	2.273	22,8
4	DEC	2.743	2.023	35,0
5	Sperry	2.552	2.270	12,4
6	Burroughs	2.478	2.442	1,5
7	Honeywell	1.634	1.453	12,5
8	Hew.-Pack.	1.577	1.147	37,5
9	Xerox	770	570	35,1
10	Memorex	686	658	4,3
	Skupaj 10	39.438	33.710	17,0
	Skupaj 100	55.626	46.220	20,4
	Prvih 10 v 100	70,9 %	72,9 %	

Tabela 2

Povečanje dohodka (v 100 milijonih dolarjev)

		povečanje dohodka 1980	rast dohodka 1980 (%)
1	IBM	3.029	16,5
2	DEC	712	35,0
3	Control Data	518	22,8
4	Hew.-Pack.	430	37,5
5	NCR	312	12,3
6	Sperry	282	12,4
7	Wang Labs	271	66,1
8	Xerox	200	35,1
9	Honeywell	182	12,5
10	Texas Inst.	164	41,2
11	Comp.Sciences	145	34,8
12	Data General	133	24,7
13	Storage Technology	124	25,9
14	Prime Computer	115	75,0
15	Apple	105	175,1
16	Teletype	105	72,4
	Skupaj 16	6.764	20,7
	Skupaj 100	9.344	20,4

Tabela 3

20 najboljših v povečanju dohodka (v milijonih dolarjev)

		skupna rast (%)	rast v ZDA (%)	rast v inoz. (%)	dohodek 1980	dobiček 1980
1	Sanders Assoc.	208,5	91,5	n.p.	145,0	49
2	Apple	175,1	163,4	224,7	165,2	47
3	Philips Inf. Sys.	100,0	100,0	n.p.	50,0	98
4	Tandem	93,9	58,7	179,4	128,8	53
5	Intergraph	91,3	80,1	153,7	56,5	90
6	Dysan	86,1	79,9	127,7	62,9	85
7	Computervision	85,5	72,7	108,3	191,1	41
8	Paradyne	83,2	74,0	108,9	75,9	77
9	Prime	75,0	59,9	95,8	267,6	27
10	Teletype	72,4	62,1	n.p.	250,0	29
11	CPT	68,9	44,7	157,7	76,4	76
12	Wang Labs	66,1	68,8	61,5	681,8	11
13	Lanier	64,1	60,6	129,7	128,0	54
14	Triad Systems	61,0	61,0	n.p.	60,2	87
15	Anacomp	60,1	60,1	60,0	57,0	89
16	Commodore Int.	54,1	-13,2	105,2	98,7	66
17	Applicon	51,4	35,2	136,6	68,5	82
18	Auto-trol Technology	51,3	58,5	27,1	50,8	97
19	AM International	49,0	49,0	49,0	98,8	65
20	Printronix	48,8	37,7	93,0	48,9	99

n.p. - ni primerno

Največjih 10 podjetij je doseglo dohodek 39,4 milijarde dolarjev s stopnjo rasti 17 % in z 71 % prihoda 100 podjetij. Podjetje IBM je še vedno vodilno, DEC se je povzpel s šestega mesta na četrto ter tudi Amdahl se je za 97 milijonov dolarjev večji dohodek, da bi zasedel drugo mesto. Burroughs je padel iz drugega mesta na šesto, med prvih deset pa se je prebil Xerox (namesto Data General).

IBM je pridelal največje povečanje svojega dohodka, in sicer več kot 3 milijarde dolarjev (glej tabelo 1) v enem letu, pri DEC-u je znesek to povečanje 712 milijonov dolarjev (glej tabelo 2). Le 16 podjetij (tabela 2) je doseglo povečanje dohodka, ki je večje od 100 milijonov dolarjev.

Čeprav je bilo leto 1980 uspešnejše od poslovnega leta 1979, je v letu 1979 osem podjetij doseglo povečanje dohodka med 100 %, v letu 1980 pa so bila taka podjetja le tri (Apple dvakrat zaporedoma, Sanders Ass. in Philips Information Syst.). Ti podatki so zbrani v tabeli 3. Iz te tabele je razvidno, da so bila mikroračunalniška podjetja zelo uspešna (večina podjetij v tabeli 3 proizvaja mikroračunalnike).

Med podjetji, ki so najslabše poslovala, se nahajajo podjetja Burroughs (1,5), ITT (2,0), National Semi-

Tabela 4

10 najboljših v operativnem dobičku (v milijonih dolarjev)

		1980	1979	spremembe (%)
1	IBM	5.231	4.649	12,5
2	DEC	434	327	32,6
3	NCR	390	348	12,0
4	Hewlett-Packard	261	184	52,7
5	Sperry	255	206	23,7
6	Control Data	238	176	35,1
7	Honeywell	186	152	21,9
8	Burroughs	145	456	-68,2
9	Wang Labs	113	70	64,7
10	Data General	105	92	14,6
	Skupaj 10	7.379	6.660	10,8
	Skupaj 100	5.500	7.715	10,2

Tabela 5

10 najboljših: kapitalni izdatki (v milijonih dolarjev)

		1980	1979	spremembe (%)
1	IBM	1.985	1.548	28,2
2	DEC	321	125	156,4
3	Control Data	296	208	42,8
4	NCR	156	115	36,2
5	Hewlett-Packard	148	115	28,7
6	Burroughs	147	100	47,0
7	Sperry	117	75	56,7
8	Wang Labs	96	65	48,9
9	Storage Technology	76	56	36,0
10	Automatic Data Proc.	70	65	7,7
	Skupaj 10	3.412	2.470	38,1
	Skupaj 100	4.077	2.975	37,1

Tabela 6

10 najboljših: izdatki za R in R (v milijonih dolarjev)

		1980	1979	sprememba %
1	IBM	1.277	1.125	13,5
2	DEC	217	155	39,9
3	Sperry	216	189	14,1
4	NCR	201	171	17,5
5	Control Data	183	149	22,4
6	Burroughs	175	152	15,1
7	Honeywell	150	117	28,2
8	Hewlett-Packard	139	103	35,0
9	Data General	68	54	26,2
10	Amdahl	63	42	49,3
	Skupaj 10	2.688	2.257	19,1
	Skupaj 100	3.713	3.057	21,5

Tabela 7

10 najboljših: zaposleni (v tisočih)

		1980	1979	sprememba %
1	IBM	278	270	2,9
2	NCR	66	65	1,6
3	DEC	60	50	21,0
4	Burroughs	57	57	1,4
5	Control Data	49	48	1,4
6	Sperry	47	46	2,8
7	Honeywell	29	29	1,8
8	Hewlett-Packard	28	25	12,0
9	Computer Sciences	15	13	10,6
10	Data General	14	14	4,8
	Skupaj 10	643	616	4,5
	Skupaj 100	863	806	7,1

Tabela 8

10 najboljših: procesiranje teksta (v milijonih dolarjev)

		PT 1980	sprem. %	CD 1980	PT/CD %	mesto med 100
1	Wang Labs	252,3	112,0	681,8	37,0	11
2	Lanier	110,1	48,6	128,0	86,0	54
3	DEC	82,3	102,6	2.743,3	3,0	4
4	Xerox	69,3	35,1	770,0	9,0	9
5	Raytheon	67,5	37,8	225,0	30,0	34
6	Exxon	66,2	31,9	86,0	77,0	71
7	CPT	65,7	68,9	76,4	86,0	76
8	Burroughs	49,6	1,5	2.478,0	2,0	6
9	Philips Info Systems	46,0	100,0	50,0	92,0	98
10	NBI	40,1	95,5	43,2	93,0	109

PT - procesiranje teksta, CD - celoten dohodek

conductor (6,5), Ampex (12,2), NCR (12,3) in Honeywell (12,5) (v oklepaju je prirastek dohodka v % glede na prejšnje leto).

Merilo zdravja (ali konjunkture) industrije je prav gotovo rast dobička (ali točneje rast gotovinskega pretoka iz operacij), ki se lahko pretaka v fonde za kapitalne investicije ter v izdatke za raziskave in razvoj; vse to bistveno vpliva na prihodnjo rast podjetja oziroma na obseg proizvodnje. Operativni dobiček, prikazan v tabeli 4, je določen kot dohodek iz proizvodov in uslug; od tega so odšteti stroški prodanih izdelkov, izdatki za R in R, prodajo in administracijo. Tak dobiček je merilo dobičkonosnosti in večletne primerjave tega merila so bistvene.

Že v letu 1979 se je povečala kapitalna poraba za 30 %; v letu 1980, ko je IBM upočasnil svojo rast, je znašala ta poraba še vedno 28 % (glej tabelo 5), pri 100 podjetjih pa v povprečju 37 %. Celotna kapitalna poraba je dosegla več kot 4 milijarde dolarjev. Iz tabele 5 se vidi, da IBM pri tem še krepko vodi s skoraj 2 milijardami kapitalne porabe. Podobne težnje rasti porabe so opazne pri proizvajalcih miniračunalnikov in procesorjev teksta (DEC, HP, Wang).

Računalniška industrija je bila zaradi težkih tehnoloških pogojev, kratkih produktnih ciklov in hitrih tržnih sprememb dokaj dober investitor v razvoj in raziskave. R in R vključuje finančno izdatke za nenajavljene nove proizvode in usluge v samem podjetju. Od 18 % izdatkov za R in R v letu 1979, je bilo v letu 1980 doseženih že 21 % ali 3,7 milijarde (glej tabelo 6). Izdatki za R in R so se glede na prodajo povečali iz 7,5 % na 7,6 %. Največje povečanje izdatkov za R in R je bilo doseženo na področju pisarniških sistemov (procesiranje teksta), in sicer do 102 %. V tem desetletju se pričakuje tudi znatno povečanje izdatkov za R in R na področju programske opreme.

Povečanje zaposlenih v računalniški industriji ZDA v letu 1980 je znašalo 57.000 ali 7 % v primerjavi z letom 1979, kot kaže tabela 7. Prvih 10 podjetij je zaposlovalo kar 74 % vse delovne sile, pri vseh 10 pa so dohodki rasli hitreje kot zaposlenost. To pomeni, da je produktivnost zaposlenih naraščala, in sicer s 53.600 dolarjev na zaposlenega v letu 1979 na 60.100 dolarjev na zaposlenega v letu 1980.

Dohodki na področju materialne opreme za procesiranje tekstov so narasli za 64 %, in sicer na vrednost 881 milijonov dolarjev v letu 1980. Vodilni proizvajalec je bilo podjetje Wang Labs, ki je svoj dohodek povečalo za 112 % oziroma na vrednost 252 milijonov dolarjev (glej tabelo 8).

Ta bežen pregled poslovanja ameriške računalniške industrije v letu 1980 kaže, da je v vzponu še vedno mikroračunalniški proizvodni segment in da so pisarniški sistemi hitro rastoče področje uporabe oziroma plasmaja. Hkrati narašča tudi pretok dohodkov v raziskovalne in razvojne dejavnosti, zaposlovanje pa narašča zmerno ob povečevanju produktivnosti zaposlenih.

A. P. Železnikar

Prihodnost procesorja 8086

V preteklih mesecih je Intel izzval pozornost z 32 bitnim mikroprocesorjem iAPX 432. O novih 16 bitnih procesorjih pa ni bilo nobenih informacij. Sedaj pa je Intel napovedal predstavitev dveh 16 bitnih mikroprocesorjev, ki sta ukazno kompatibilna s procesorjema 8086/8088.

Projekt iAPX 186 je hardwarska razširitev 8086 z enako arhitekturo in enakimi perifernimi integriranimi vezji. iAPX 286 bo imel bistveno razširjen nabor ukazov, ki bo olajševal implementacijo kompleksnih operacijskih sistemov in visokih jezikov. Ta mikroprocesor razpolaga s fizičnim naslovnim prostorom obsega 16 Mzlogov in z virtualnimi velikosti 1000 Mzlogov. Enota za upravljanje s pomnilnikom bo integrirana v procesorskem čipu.

Drago Novak

Uporaba enostranskih disket za dvostranski zapis in branje

Pri uporabi dvostranskih disketnih programov je bilo ugotovljeno, da je pri t. im. enostranskih disketah (IBM) možno čitati tudi na drugi strani diskete. Kasnejši poskus je prikazal, da velja to tudi za druge diskete (Ampex, DEC). S posebnim programom formaliramo (ali inicializamo) tudi drugo stran (v dvostranskem disketnem programu), nakar lahko začnemo vpisovati sektorje oziroma uporabljati tudi drugo stran diskete.

Ob tem nastane vprašanje, zakaj so dvostranske diskete dražje od enostranskih. Proizvajalec, ki prodaja dvostranske diskete, garantira kvaliteto diskete na obeh straneh, oziroma preizkusi obe strani diskete. Pri enostranskih disketah je garantirana samo ena stran. Enostranske diskete so tudi vse tiste, ki niso zadovoljile preizkusu na obeh straneh ali so bile preizkušene samo na eni strani.

Drugo stran diskete naj bi zaradi tega uporabljali s previdnostjo, lahko pa si naredimo tudi svoj preizkusni program, ki preizkusi disketo po njeni površini. Na ta način lahko ugotovimo slabe sektorje, ki jih iz uporabe izločimo. Na drugi strani enostranske diskete zapisujemo praviloma le nekritične podatke oziroma programe.

A. P. Železnikar

Tudi IBM in HP pod $ 10.000

Največja računalniška družba IBM je vstopila v donosno tržišče računalnikov s prodajnimi cenami od 6.500 do 10.000 dolarjev s sistemom System/23 Datamaster. Prodajajo ga za 9.830 dolarjev, s tiskalnikom in dvema disketnima pogonoma (8 col). Za podoben korak se je odločil tudi drugi največji dobavitelj miniračunalnikov, HP.

Zdaj, ko so se nekateri glavni proizvajalci računalnikov vključili v proizvodnjo cenenih sistemov, bi se naj na tem področju pričela huda bitka za nadvlado. Vendar ni verjetno, da bi to komu tudi uspelo, niti družbi IBM. Prodaja sistemov, ki stanejo pod 20.000 dolarjev, bo od 1980 do 1984 predvidoma rastla letno po stopnji 33,5 % in bo leta 1984 znašala 50 % vrednosti prodaje vseh sistemov.

Namizni sistem System/23 je samostojna enota, ki bazira na 8-bitnem mikroprocesorju Intel 8085. Na razpolago so računalniške konfiguracije z 32 - 128 K zlogov pomnilnika. Pomnilnike dobavljajo različni proizvajalci. Računalnik s 64 K pomnilnika in 2,8 M prostora na disketah stane 7.050 dolarjev. Sistem ima tudi 112 K pomnilnika tipa ROM. Pod standardno programsko opremo spada operacijski sistem in visok programirni jezik BASIC. Nabavijo se seveda lahko še drugi programski paketi. Sistem za procesiranje teksta rabi 64 K pomnilnika in še dodaten modul, ki krmili prikazovalnik in omogoča funkcije kot navpično in vodoravno segmentiranje, tabuliranje, robljenje, ... Programska oprema za procesiranje teksta stane 500 dolarjev, modul pa 600 dolarjev.

Podobno kot sistem System/23, tudi namizni sistem HP 125 združuje funkcije majhnega poslovnega računalnika, osebnega računalnika in sistema za procesiranje teksta. Lahko služi tudi kot inteligentni terminal k poslovnemu računalniku iz serije HP 3000 in s tem omogoča tudi manjšim uporabnikom dostop do velikih podatkovnih baz.

Oba ponujena sistema - System/23 in HP 125 - sta po ceni in zmogljivostih zelo podobna tistim, ki so jih v zadnjem času že predstavile druge velike firme (DEC, Datapoint, Xerox). V ZDA bodo te računalnike prodajali v trgovinah z malimi poslovnimi sistemi in preko že utrjenih prodajnih mrež za velike sisteme.

System/23 je še vedno predrag, da bi lahko postal osebni računalnik, ki so ga nekateri pričakovali od družbe IBM. Sedanji mali sistemi velikih proizvajalcev še vedno ne konkurirajo proizvodom vodilnih na področju osebnih računalnikov (Apple, Tandy, Commodore). Analitiki pa napovedujejo, da bo IBM kmalu predstavil osebni računalnik, ki bo stal od 3.000 do 4.000 dolarjev.

B. Blatnik

SREČANJA

6-9 oktober, Copenhagen, Danska
9th European Conference on Computer Measurement

Org.European Computer Measurement Association
Informacije:Scott N. Yasler, President, ECOMA, Scheuchzer-strasse 5, CH-8006 Zurich, Switserland

7-9 oktober,Tokyo, Japonska
11th International Symposium on Industrial Robots

ORganisator in infoemacije:Society of Biomechanics and Japan Industrial Robot Association

13-16 oktober,Couvent Royal de Saint Maxim, France
Information Systems 81

Org.INRIA, ADI
Informacije:Najah Naffah, INRIA, op105, 78150 Le Chesnay France

19-23 oktober,Munich, ZRNemčija
Third Conference of the European Cooperationi in Informatics

ORg.Gesellschaft fur Informatik FRG
Informacije:A.J.W.Duijvestijin,POB 217, 7500 Enschede The Netherlands

26-29 oktober,Amsterdam, Nizozemska
International Symposium on Algorithmic Languages
ORg.IFIP TC2.
Informacije:Algoritmic Languages 81,Mathematical Center, Kruislaan 413, 1098 SJ Amsterdam , The Netherlands

26-30 oktober, Lyon, Francija
Nato Symposium on Artificial and Human Intelligence

Org.Human Factors Subcommittee of NATO
Informacije:Alick Elithorn, The RoyalFree Hospital, Pond Street, London NW3, 2QG, England

27-29 oktober,London, Velika Britanija
TESTMEX,Conference and exbithion devoted to electronic test and measurement instrumentation

Org. in informacije:Trident International Exhbitions Ltd. 21 Plymouth Road, Tavistoak, Devon PL19 8 AU, UK

10-11 november, London, Velika Britanija
Survey data processing now:microcomputers,data bases, data presentation and display techniques

Org. in informacije:A.Morris, State MR Ltd.,Walkden House Melton Street, Euston Road, London NW1 2EB, UK

17-19 november,London, Velika Britanija
2nd International Symposium on technical diagnostics

Org. in informacije:Institute of Measurement and Control 20 Peel Street, London W8 7PD, UK

18-21 november, Varna, Bugarija
COMPCONTROL 81

ORg. in informacije: Union of Mechanical Engineering, Rakovski St. 108, Sofia - 1000, Bulgaria

14-18 december, Versaille, Francija
5th International Conference on Computing Methods

Org.INRIA
Informacije:INRIA, Service des Relations Exterieures, Domain de Voluceau, B.P. 105, 78153, Le Chesnay, France

1982 leto

27-29 januar,Eindhoven, Nizozemska
Microelectronica - 2nd European microelectronic Congress

Org. in informacije: Microelectronica, PO Box 428, Los Altos CA 94022, USA

2-4 februar, London, Velika Britanija
Electronic OEM Assemblies 82

Org. in informacije:Trident International Exhibitions Ltd. 21 Plymouth Road, Tavistook, Devon, PL19 8AU, UK

24-26 februar, London, Velika Britanija
Microsysteme 82
Org. in informacije: IPC Exhibitions Ltd., Surrey House, 1 Throwley Way Sutton, SurreySM1 4QQ, UK

30 marec - 1 april, Brighton, Velika Britanija
5th International Conference on Computers in design engineering

Org. in informacije:Judy Vare, Conference Secretary, IPC Science and Technology Press Ltd. POB 63, Westbury House, Bury Street, Guildford GU2 5BH, UK

6-8 april, Torino, Italija
5th International Symposium on Programming

Org. Istituto di Scienze dell' Informazione
Informacije: S. Ronchi della Rocca, Istituto di Scienze dell' Informazione, C.M.D' Agazlio 42 -10125 Torino, Italy

24-28 maj, Berlin, ZR Nemčija
9th World Conference IMEKO

Org. in informacije: Gesellschaft Mess und Regelungstechnik Graf-Recke Strasse 84 -B.P. 1139, D 4000 Dusseldorf 1, BDR

16-18 maj, Leningrad, ZSSR
PROLAMAT 81

Org. in informacije: Leningrad Researche Computer Centre Mendeleyevskaya Linia 1, USSR Academy of Science, 199164 Leningrad, USSR

16-18 juni, Bruxelles, Belgija
7th International Conference on Dynamic Systems

Org. Universite de Paris Dauphine
Informacije: B. Paulre, Universite de Paris Dauphine, place de Mal. de Lattre de Tassigny, 75725 Paris, France

29 juni - 2 juli, Toulouse, Francija
3rd IFAC Symposium on Control of Distributed Parameter Systems

Org.: IFAC, INRIA
Informacije: INRIA, Service de Relations Exterieures, Domaine de Voluceau, B.P. 105, 78153 Le Chesnay, France

22-24 juni, Jerusalem, Izrael
2nd International Conference on Data Bases

Org.: Hebrew Universityand Northwestern University
Informacije: Michael Hanani, Univereityof the Negev, Beer Sheba, Izrael

5-10 juli, Praha, ČSSR
9th Conference on Computational Linguistics

Org.: International Committee on Computational Linguistics
Informacije: COLING 82, MFF UK, Linguistics, Malostranske n. 25, 118 00 Praha 1, ČSSR

11-15 juli, Warszawa, Poljska
3rd IFAC/IFORS Symposium on Large Scale Systems

Org.: IFAC/IFORS.
Informacije: Z. Nahorski, System Research Institute, Polish Academy of Sciences, ul. Newelska 6 - 01 447 Warszawa, Poland

11-16 juli, Washington, ZDA
6th International Conference on Computers in Chemical Researche and Education

Org. ICCCRE
Informacije: R. Heller, EPA, MIDSD, PM-218, 401 M Street, S.W. Washington, DC 20460, USA

26-30 juli, Heidelberg, Z R Nemčija
APL 82

Org. in Informacije: K. Waller - APL 82, postfach 101248 - D 6900 Heilderberg 1, BDR

24-27 avgust, Bordeaux, Francija
Advances in Production Management Systems

Org.: IFIP, W.G.5.7, AFCET, Universite de Bordeaux
Informacije: Guy Doumeingts, APMS 82, G.R.A.I. - Universite de Bordeaux I, 33405 Talence Cedex, France

12-16 september, Berlin, Z R Nemčija
International Conference for Data Processing and Information Technology

Org.: AMK Berlin and ACM European Region
Informacije: E. Fuchs, Austellungs-Messe-Kongress, GmbH, Messedamm 22, D-1000 Berlin 19, BDR

1983 leto

19-23 september, Paris, Francija

IFIP

Org.: IFIP in cooperation with SICOB
Informacije: IFIP Secretariat, 3 rue du Marche, CH-1204 Geneva, Switzerland

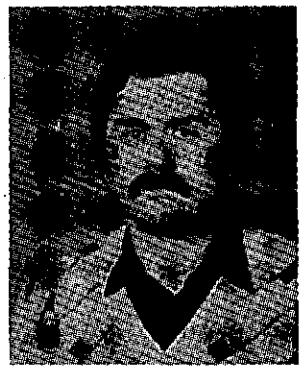
RAZSTAVE IN SEJMI

6-9 oktober, 1981, Birmingham, Velika Britanija
Design Engineering Show and FactoryManagement & Maintenance Engineering Show

21-22 oktober, London, Velika Britanija
Computer-aided Manufacturing&Productivity

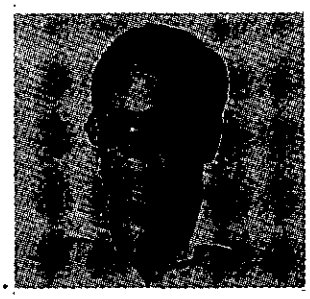
21-25 oktober, 1981, Dortmund, Z R Nemčija
Elektrotechnik 81
27-31 oktober, Berlin, Z R Nemčija
Elektrotechnische Fachschau

AVTORJI IN SODELAVCI



JANEZ KOŽUH je bil rojen v Mariboru 15. 7. 1949. Leta 1975 je diplomiral na Fakulteti za elektrotehniko v Ljubljani, smer avtomatika in se zaposlil v Iskri tehničnem razvoju TOZD-a Računalniki. V začetku je delal na področju mikroračunalništva in je sodeloval pri razvoju sistema ISKRADATA 1680. Od leta 1978 dela na razvoju strojne opreme računalniških sistemov ISKRADATA 18-20 in ISKRADATA 19.

NOVAKOVIČ ŽARKO je bil rojen 28. junija 1952. Končal je drugo in tretjo stopnjo (magisterij) na elektrotehniški fakulteti University of Maryland, ZDA, 1974 oziroma 1976 leta. Na isti fakulteti je bil asistent ter predavatelj v obdobju 1975 - 1978. Od 1980 dela v tehničnem razvoju, ISKRA Telekomunikacije, Kranj na razvoju programsko krmiljenega digitalnega sistema telefonskih central ISKRA 2000 kot samostojni razvijalec.



MARKO ROGAČ rojen 4. 8. 1947 v Kranju. Od leta 1976 je zaposen v tehničnem razvoju Iskra Elektromehanika Kranj, TOZD Računalniki. Diplomiral je na Fakulteti za elektrotehniko v Ljubljani. Sodeloval pri razvoju mikroračunalnika ID 1680, potem se je vključil v delo na 16-bitnem mikroračunalniku.

Lado Petrnelj je bil rojen v Škofji loki 9.7.1948. Leta 1973 je diplomiral na Fakulteti za elektrotehniko v Ljubljani, smer Avtomatika. Magistrsko nalogo je zagovarjal leta 1975, nakar se je zaposlil v Iskri, TOZD Računalniki, Tehnični razvoj. Delal je na področju mikroračunalništva ter sodeloval pri razvoju sistema Iskradata 1680. Od leta 1978 dela na razvoju sistemske programske opreme računalnikov Iskradata 18-20 in Iskradata 19.

HAFNER DAMIJAN je bil rojen 22. 3. 1957 v Kropi. Leta 1980 se je zaposlil v tehničnem razvoju Iskra Elektromehanika Kranj, TOZD Računalniki in diplomiral na Fakulteti za elektrotehniko v Ljubljani, smer računalništvo. V začetku je delal na povezavi mikroračunalnika ID 1680 z miniračunalnikom ID C-18/20, potem se je vključil v delo na 16-bitnem mikroračunalniku.

NAVODILO ZA PRIPRAVO ČLANKA

Avtorje prosimo, da pošljejo uredništvu naslov in kratek povzetek članka ter navedejo približen obseg članka (število strani A 4 formata). Uredništvo bo nato poslalo avtorjem ustrezno število formularjev z navodilom.

Članek tipkajte na priložene dvokolonske formularje. Če potrebujete dodatne formularje, lahko uporabite bel papir istih dimenzij. Pri tem pa se morate držati predpisanega formata, vendar pa ga ne vrišite na papir.

Bodite natančni pri tipkanju in temeljiti pri korigiranju. Vaš članek bo s foto postopkom pomanjšan in pripravljen za tisk brez kakršnihkoli dodatnik korektur.

Uporabljajte kvaliteten pisalni stroj. Če le tekst dopušča uporabljajte enojni presledek. Črni trak je obvezen.

Članek tipkajte v prostor obrobljen z modrimi črtami. Tipkajte do črt - ne preko njih. Odstavek ločite z dvojnim presledkom in brez zamikanja prve vrstice novega odstavka.

Prva stran članka:
a) v sredino zgornjega okvira na prvi strani napišite naslov članka z velikimi črkami;
b) v sredino pod naslov članka napišite imena avtorjev, ime podjetja, mesto, državo;
c) na označenem mestu čez oba stolpca napišite povzetek članka v jeziku, v katerem je napisan članek. Povzetek naj ne bo daljši od 10 vrst.
d) če članek ni v angleščini, ampak v katerem od jugoslovanskih jezikov izpustite 2 cm in napišite povzetek tudi v angleščini. Pred povzetkom napišite angleški naslov članka z velikimi črkami. Povzetek naj ne bo daljši od 10 vrst. Če je članek v tujem jeziku napišite povzetek tudi v enem od jugoslovanskih jezikov;
e) izpustite 2 cm in pričnite v levo kolono pisati članek.

Druga in naslednje strani članka:
Kot je označeno na formularju začnite tipkati tekst druge in naslednjih strani v zgornjem levem kotu,

Naslovi poglavij:
naslove ločuje od ostalega teksta dvojni presledek.

Če nekaterih znakov ne morete vpisati s strojem jih čitljivo vpíšite s črnim črnilom ali svinčnikom. Ne uporabljajte modrega črnila, ker se z njim napisani znaki ne bodo preslikali.

Ilustracije morajo biti ostre, jasne in črno bele. Če jih vključite v tekst, se morajo skladati s predpisanim formatom. Lahko pa jih vstavite tudi na konec članka, vendar morajo v tem primeru ostati v mejah skupnega dvokolonskega formata. Vse ilustracije morate (nalepiti) vstaviti sami na ustrezno mesto.

Napake pri tipkanju se lahko popravljajo s korekcijsko folijo ali belim tušem. Napačne besede, stavke ali odstavke pa lahko ponovno natipkate na neprozoren papir in ga pazljivo nalepite na mesto napake.

V zgornjem desnem kotu izven modro označenega roba oštevilčite strani članka s svinčnikom, tako da jih je mogoče zbrisati.

Časopis INFORMATICA
Uredništvo, Parmova 41, 61000 Ljubljana

Naročam se na časopis INFORMATICA. Predplačilo bom izvršil po prejemu vaše položnice.

Cenik: letna naročnina za delovne organizacije 500,00 din, za posameznika 200,00/100,00/50,00 din

Časopis mi pošiljajte na naslov ☐ stanovanja ☐ delovne organizacije.

Priimek.....

Ime.....

Naslov stanovanja

Ulica.....

Poštna številka_____ Kraj.....

Naslov delovne organizacije

Delovna organizacija.....

.....

Ulica.....

Poštna številka_____ Kraj.....

Datum..... Podpis:

.....

INSTRUCTIONS FOR PREPARATION OF A MANUSCRIPT

Authors are invited to send in the address and short summary of their articles and indicate the oppproximate size of their contributions (in terms of A 4 paper). Subsequently they will receive the outor's kits.

Type your manuscript on the enclosed two-column-format manuscript paper. If you require additional manuscript paper you can use similar-size white paper and keep the proposed format but in that case please do not draw the format limits on the paper.

Be accurate in your typing and through in your proof reading. This manuscript will be photographically reduced for reproduction without any proof reading or corrections before printing.

Use a good typewriter. If the text allows it, use single spacing. Use a black ribbon only.

Keep your copy within the blue margin lines on the paper, typing to the lines, but not beyond them. Double space between paragraphs.

First page manuscript:

a) Give title of the paper in the upper box on the first page. Use block letters.
b) Under the title give author's names, company name, city and state - all centered.
c) As it is marked, begin the abstract of the paper. Type over both the columns. The abstract should be written in the language of the paper and should not exceed 10 lines.
d) If the paper is not in English, drop 2 cm after having written the abstract in the language of the paper and write the abstract in English as well. In front of the abstract put the English title of the paper. Use block letters for the title. The lenght of the abstract should not be greater than 10 lines.
e) Drop 2 cm and begin the text of the paper in the left column.

Second and succeeding pages of the manuscript:
As it is marked on the paper, begin the text of the second and succeeding pages in the left upper corner.

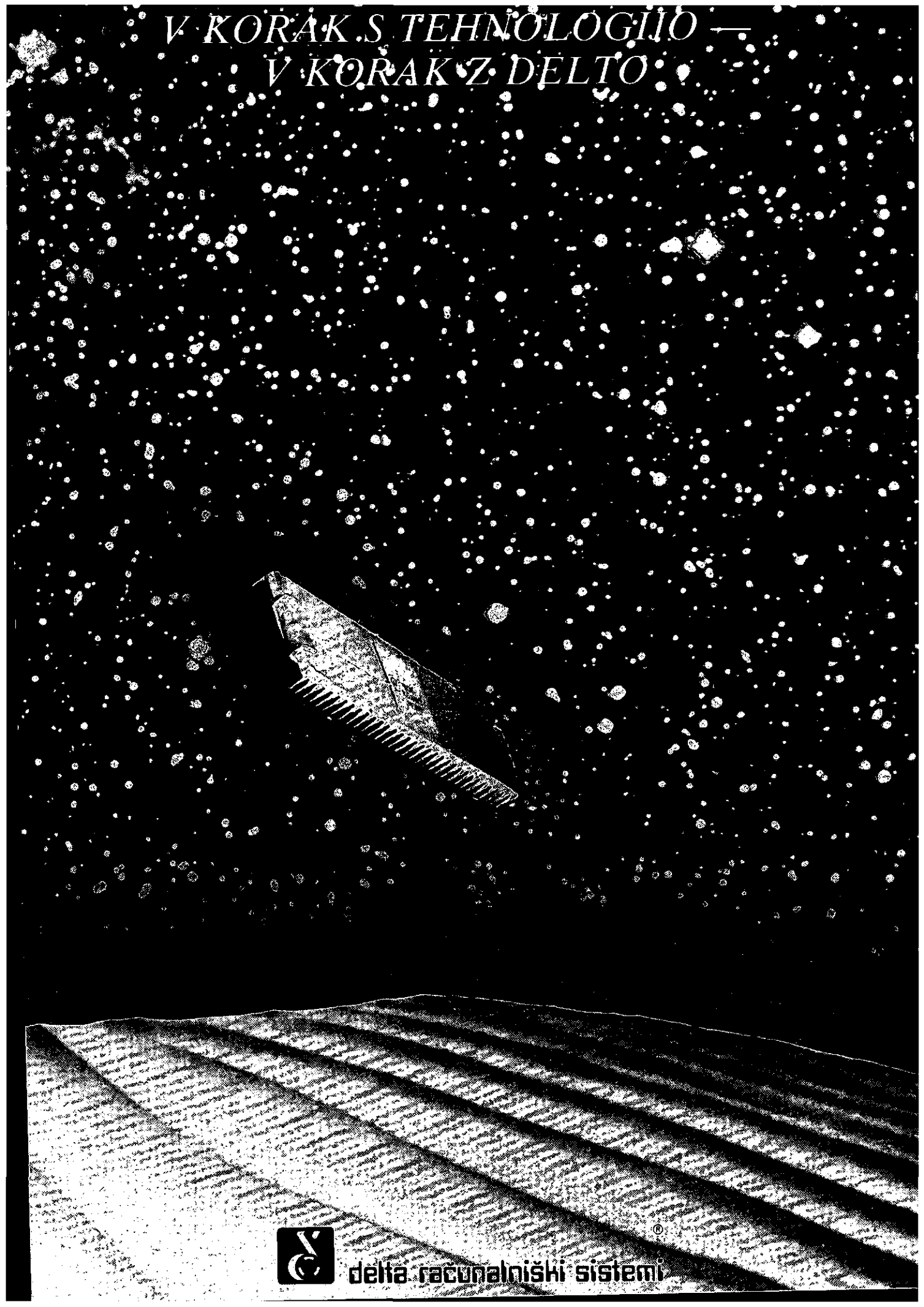
Format of the subject headings:
Headings are separated from text by double spacing.

If some characters are not available on your typwriter write them legibly in black ink or with a pencil. Do not use blue ink, because it shows poorly.

Illustrations must be black and white, sharp and clear. If you incorporate your illustrations into the text keep the proposed format. Illustration can also be placed at the end of all text material provided, however, that they are kept within the margin lines of the full size two-column format. All illustrations must be placed into appropriate positons in the text by the author.

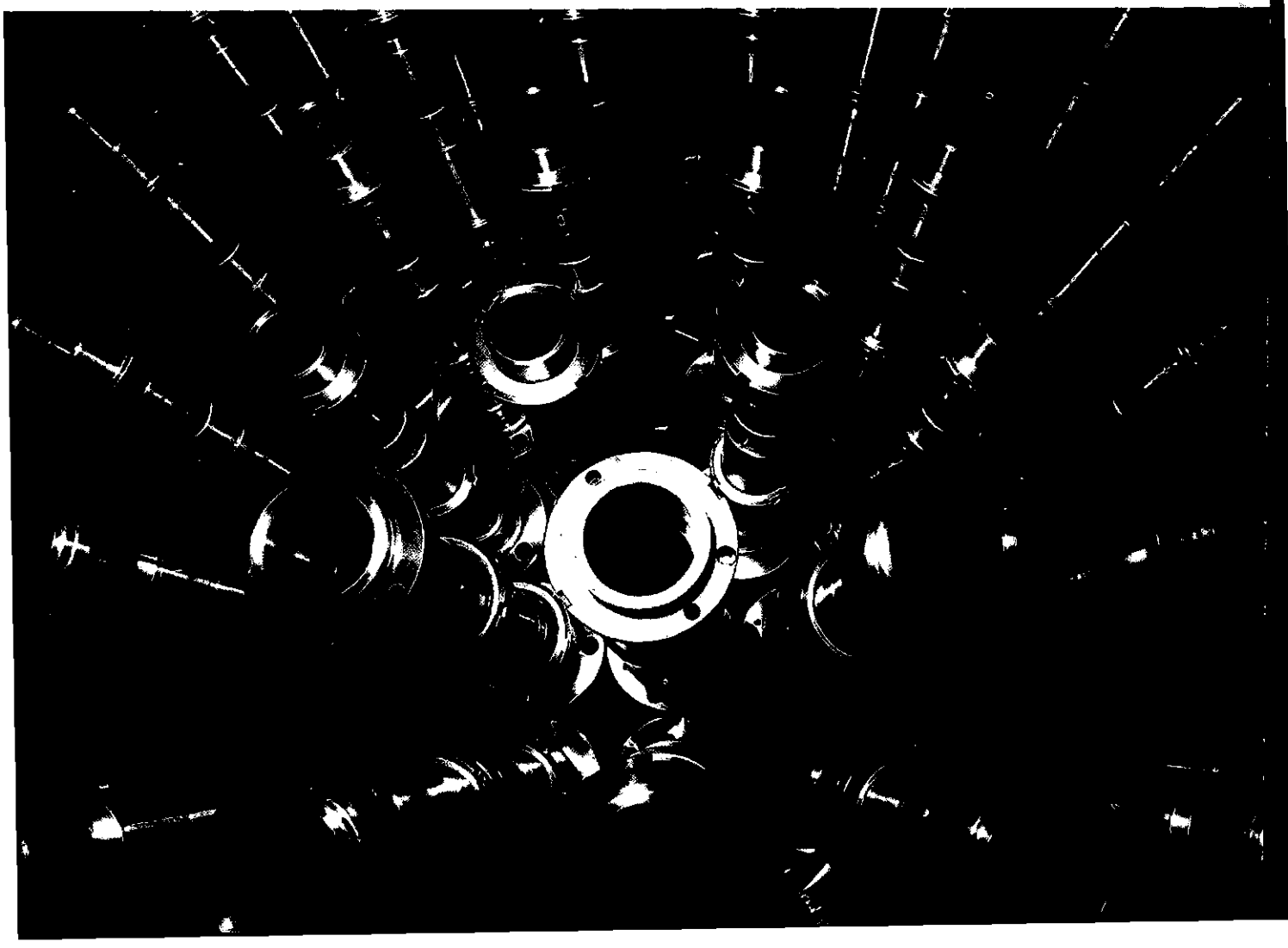
Typing errors may be corrected by using white correction paint or by retyping the word, sentence or paragraph on a piece of apaque, white paper and pasting it nearly over errors

Use pencil to number each page on the upper-right-hand corner of the manuscript, outside the blue margin lines so that the numbers may be erased.


V KORAK S TEHNOLOGIJO —
V KORAK Z DELTO
delta računalniški sistemi

Delavci DO DELTA proizvajamo najpopolnejšo jugoslovansko družino računalnikov, katera obsega celotno območje od mikro računalnikov do največjih 32-bitnih sistemov.
Poseben pomen dajemo aplikacijski programski opremi. Ob izbiri segmentov je bila naša skrb posvečena povečanju produktivnosti in čimvečjemu prihranku energije ter surovin. Programski moduli za področja procesne kontrole, planiranja in upravljanja proizvodnje, ter finančnega poslovanja, predstavljajo integralen pristop v izgradnji informatizirane proizvodne delovne organizacije. Naši računalniki so narejeni tako, da niso element prestiža delovnih organizacij, ki jih kupujejo, temveč so orodje razvojnega inženirja, projektanta, delavca v skladišču in drugih. S takim načinom dela vstopa DELTA skupaj s svojimi uporabniki v informatizirano družbo prihodnosti . . .

Če želite več informacij o DELTI, pišite na naslov: ELEKTROTEHNA — DO DELTA, Služba za komuniciranje s tržiščem, Parmova 41, 61000 Ljubljana.



CENIK OGLASOV

Ovitek - notranja stran (za letnik 1981)

2 stran	28.000 din
3 stran	21.000 din

Vmesne strani (za letnik 1981)

1/1 stran	13.000 din
1/2 strani	9.000 din

Vmesne strani za posamezno številko

1/1 stran	5.000 din
1/2 strani	3.300 din

Oglasi o potrebah po kadrih (za posamezno številko)

2.000 din

Razen oglasov v klasični obliki so zaželjene tudi krajše poslovne, strokovne in propagandne informacije in članki. Cene objave tovrstnega materiala se bodo določale sporazumno.

ADVERTIZING RATES

Cover page (for all issues of 1981)

2nd page	1300 $
3rd page	1000 $

Inside pages (for all issues of 1981)

1/1 page	790 $
1/2 page	520 $

Inside pages (individual issues)

1/1 page	260 $
1/2 page	200 $

Rates for classified advertizing:

each ad	66 $

In addition to advertisment, we welcome short business or product news, notes and articles. The related charges are negotiable.

SLOVENSKO DRUŠTVO I N F O R M A T I K A , Parmova 41, 61000 Ljubljana

I Z J A V A

Podpisani ..... izjavljam, da želim vstopiti v Slovensko društvo I N F O R M A T I K A in da sprejemam Statut društva (objavljen v časopisu Informatica štev. 1, stran 67, letnik 1981).

P o d p i s :

.....

Ime in priimek: .....

Točen naslov: .....

Datum: .....