tanulmányok

69/1977

MTA Számítástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

OPERÁCIÓS RENDSZEREK ELMÉLETE
III. VISEGRÁDI TÉLI ISKOLA

ISBN 963 311 049 1
ISSN 0324-2951

Tanulmányok 69/1977.

Felelős kiadó:

Dr VÁMOS TIBOR

MTA Számítástechnikai és Automatizálási Kutató Intézet
MTA Számítástudományi Bizottsága

Készült:

az ORSZÁGOS MŰSZAKI ÉS DOKUMENTÁCIÓS KÖZPONT
házi sokszorositójában
F.v.: Janoch Gyula

A konferenciát a "Számítástechnika tudományos kérdései" c.
többoldalú akadémia együttmüködés keretében rendeztük.

Конференция была проведена в рамках
многостороннего сотрудничества академий
социалистических стран по проблеме
"Научные вопросы вычислительной техники"

Conference was held in the frame of the multilateral cooperation
of the academies of sciences of the socialist countries on Computer Sciences.

TARTALOMJEGYZÉK

CONTENTS

СОДЕРЖАНИЕ

A CLASS OF HIERARCHICAL PAGING ALGORITHMS

Ya.A. KOGAN
Institute of Control Sciences
Moscow, USSR

A paged virtual memory has become a common feature of large modern operating systems. The costs associated with paging have led to a great deal of research on the paging algorithms. Almost all paging algorithms discussed in the literature and implemented in computer systems are local, since they retain in main memory only those pages that were used recently in some sense. FIFO, LRU and Working Set are well known examples of such paging algorithms, which differ by the definitions of the set of most recently used pages. With local paging algorithms the rerely requested pages may replace those of real working set of a program. In this paper we define a class $\mathcal{X}$ of hierarchical paging algorithms a member of wich with properly selected parameters will be free of this disadvantage.

For paging algorithms of the class $\mathcal{X}$ the contents and ordering of the pages in main memory may be regarded as the state of the device.
Let $B_t = (i_1, i_2, \ldots, i_{m_t})$ represent the memory state at time t.
To define a paging algorithm we should indicate how B_t is transformed into B_{t+1}.

Let $B_t = \bigcup_{k=1}^{h} M_k$ where h is a parameter of the class $\mathcal{X}$, the intersection

$M_k \cap M_j = \emptyset$ for al $k \neq j$ and

$$M_1 = (i_{11}, i_{12} \ldots, i_{1m_1}) \quad , \quad M_2 = (i_{21}, i_{22}, \ldots, i_{2m_2}), \ \ldots \ ,$$
$$M_h = (i_{h1}, i_{h2}, \ldots, i_{hm_h}) \ ; \ h \leq 1, \ \sum_{i=1}^{h} m_i = m_t.$$

It should be noted that some of M_i may be empty. Now we introduce the following parameters:

It should be noted that some of M_i may be empty. Now we introduce the following parameters:

μ_τ is the maximal size of the set of the set M_τ, i.e. $|M_\tau| \leq \mu_\tau$ $\tau=1,\ldots, h$.

T_τ is the time that a page nay retain in M_τ, $\tau=1, \ldots, h$, without referencing to it. The time T_τ is measured, as usual in seconds of process (virtual) time. We suppose that $T_1 \geq T_2 \geq \ldots \geq T_h$.

t_τ is the time since the last reference to tha page $i_{\tau m_\tau} \in M_\tau$, $\tau=1,\ldots, h$. We shall use also some additional parameters the essence of which will be clear from the following definition.

Definition. Let χ_{t+1} be the page that is referenced at time $t+1$. For a paging algorithm $A \in \mathcal{X}$ the transformation rules $B_t \overset{A}{\rightarrow} B_{t+1}$ are defined by the following steps.

Step 1. Remove from M_τ those pages that are not referenced within the interval $[t+1-T_\tau , t+1]$, $\tau=1 \ldots, h$.

Step 2. Denote by $M'_\tau = (i_{\tau 1}, \ldots, i_{\tau m'_\tau})$ The ordered set of pages which is left in M_τ, $\tau= 1,\ldots,h$, after Step 1.

Let

$$B'_{t+1} = (M'_1, M'_2, \ldots, M'_h).$$

Step 3. The memory state B_{t+1} differs from B'_{t+1}

i) only by M_h if $\chi_{t+1} \notin B_{t+1}$,

ii) only by $M_\tau, 1 \leq \tau \leq h$, if $\chi_{t+1} \in M_\tau \setminus D_\tau$ where D_τ is a fixed subset of $M_\tau, 2 \leq \tau \leq h$, and $D_1 = \emptyset$,

iii) only by $M_{\tau-1}$ and M_τ if $\chi_{t+1} \in D_\tau$ where D_τ defined in ii) and $2 \leq \tau \leq h$.

Step 4. Definition of B_{t+1}

Let $0 < \alpha_\tau < 1$ and $1 \leq \ell_\tau \leq m'_\tau$, $\tau = 1, \ldots, h$, be additional parameters of the paging algorithms $A \in \mathcal{X}$.

In case i) $B_{t+1} = (M'_1, \ldots, M'_{h-1}, M''_h)$

where

$$M''_h = \begin{cases} (\alpha_{t+1}, i_{h1} \cdots, i_{hm'_h}) & \text{if } t_h \leq \alpha_h T_h \text{ and } m'_h < \mu_h, \\ (\alpha_{t+1}, i_{h1} \cdots, i_{m'_h - 1}) & \text{if } t_h > \alpha_h T_h \text{ or } m'_h = \mu_h. \end{cases}$$

In case ii)

$$B_{t+1} = (M'_1, \ldots, M'_{\tau-1}, M''_\tau, M'_{\tau+1}, \ldots, M'_h)$$

where for $\alpha_{t+1} = i_{\tau s}$

$$M''_{\tau-1} = \begin{cases} M'_\tau \\ (i_{\tau s}, i_{\tau 1}, \ldots, i_{\tau s-1}, i_{\tau s+1}, \ldots, i_{\tau m'_\tau}) & \text{if } \ell_\tau < \leq m'_\tau . \end{cases}$$

In case iii)

$$B_{t+1} = (M'_1, \ldots, M'_{\tau-2}, M''_{\tau-1}, M''_\tau, M'_{\tau+1}, \ldots, M'_h)$$

where for $\alpha_{t+1} = i_{\tau s}$

$$M''_{\tau-1} = \begin{cases} (i_{\tau s}, i_{\tau-1 1}, \ldots, i_{\tau-1 m'_{\tau-1}}) & \text{if } t_{\tau-1} \leq \alpha_{\tau-1} T_{\tau-1} \text{ and } m'_{\tau-1} < \mu_{\tau-1} \\ (i_{\tau s}, i_{\tau-1 1}, \ldots, i_{\tau-1 m'_{\tau-1}}) & \text{if } t_{\tau-1} > \alpha_{\tau-1} T_{\tau-1} \text{ or } m'_{\tau-s} = \mu_{\tau-1} \end{cases}$$

and

$$M''_\tau = \begin{cases} (i_{\tau 1}, \ldots, i_{\tau s-1}, i_{\tau s+1}, \ldots, i_{\tau m'_\tau}) & \text{if } t_{\tau-1} \leq \alpha_{\tau-1} T_{\tau-1} \\ & \text{and } m'_{\tau-1} < \mu'_{\tau-1} \\ (i_{\tau-1 m'_{\tau-1}}, i_{\tau 1}, \ldots, i_{\tau s-1}, i_{\tau s+1}, \ldots, i_{\tau m'_\tau}) \end{cases}$$

if $t_{\tau-1} > \alpha_{\tau-1} T_{\tau-1}$

or $m'_{\tau-1} = \mu_{\tau-1}$.

Thus in a general case a paging algorithm $A \in \chi$ depends on the following parameters; h , 5h numbers μ_τ, T_τ, t_τ, α_τ and ℓ_τ and h-1 of the subset D_τ.

The class χ of hierarchical paging algorithms contains a number of algorithms of practical and theoretical interest.

With $T_1=T_2=\ldots T_h=\infty$ and $D_\tau=M_\tau, 2 \leq \tau \leq h$, we have the class H introduced in [1]. There known paging algorithms, FIFO, LRU and CLIMB, are extremal members of H which are obtained for h=1, $\ell_h=\mu_h$; h=1, $\ell_h=1$ and $\mu_1=\ldots=\mu_n=1$ respectively. With h=1 and $\mu_h=\infty$ we have a Modified Working Set paging algorithm [2] which transforms for $\alpha_h=1$ in a Working Set paging algorithm.

The class χ extends considerably the control possibilities as compared with the class H . To demonstrate the advantages of the class χ, let us consider a simplest Markov model of program behaviour. For this model

$$P\{\chi_{t+1}=j \mid \chi_t=i\} = \begin{cases} p_j(1-q_i) & j \neq i \\ q_i+(1-q_i)p_i & j=i \end{cases} \qquad (1)$$

where $i,j=1,2,\ldots,n$, $0 \leq q_i < 1$, $p_i \geq 0$ and

$$\sum_{i=1}^{n} p_i = 1.$$

Denote by $F_m(A)$ the long-run page fault rate for the paging algorithm A and main memory size m . Using the correspondence between the Markov model (1) and the independent model pbtainde form (1) if $q_1=\ldots q_n=0$, we can prove that for a simplest Markov model of program behavior

$$F_m(LRU) = F_m^o(LRU) \Big/ \sum_{i=1}^{n} p_i^{v_i} \qquad (2)$$

where $F_m^o(A)$ denotes the long-run page fault rate for the independent model and $\upsilon_i=(1-q_i)^{-1}$ is the so called page duration.

Furthermore if $\upsilon_1=\ldots=\upsilon_n=\upsilon$, then for any $A \in H$

$$F_m(A) = F_m^o(A) / \upsilon.$$

Let $\sum_1^h \mu_i = m$ and consider the Markov model (1) with the parameters $p_1=\ldots=p_m=p$, $p_{m+1}=\alpha$, $p_{m+2}=\ldots=p_n=0$, $q_1=\ldots=q_m=0$ and $q_{m+1}=q$.

Suppose that $\alpha \to 0$. and $q \to 1$. Then it is easy to show that for any $A \in H$

$$F_m(A) \geq F_m(LRU).$$

For example, with $m=2$ we have

$$\lim_{\alpha\to 0, q\to 1} F_m(CLIMB) / F_m(LRU) = 7/5; \quad \lim_{\alpha\to 0, q\to 1} F_m(FIFO) / F(LRU) = 6/5.$$

Now let us consider the paging algorithm $\tilde{A} \in \chi$ defined by the following parameters: $h=2$, $T_1=T_2=\infty$, $\mu_1=m-2$ $\mu_2=2$, $D_2=i_{22}$. Then it is easy to show that

$$\sup_{p,\alpha,q} [F_m(\tilde{A}) / (\alpha / \sum_{i=1}^n p_i\upsilon_i)]$$

is independent of m, while from [1] and (2) we have

$$\sup_{p,\alpha,q} [F_m(LRU) / (\alpha / \sum_{i=1}^n p_i\upsilon_i)] = 1+ \sum_{i=1}^n 1/i$$

which is increasing logarithmically as $m \to \infty$

Distribution-free results on algorithms of the class χ may be obtained by the technique developed by the author in [1].

On the case of an independent model of program behavior explicit expression of $F_m(A)$ may be obtained for a number of paging algorithms $A \in \chi$, For example, if $T_1=T_2=\infty$, $\mu_1=\ell_1=c$, $\mu_2=\ell_2=m-c$, $D_2=i_{21}$, then

$$F_m(A) = \Sigma p_{i_1}^2 \cdots p_{i_c}^2 \, p_{i_{c+1}} \cdots p_{i_m} \sum_{j=m+1}^{n} p_{i_j} \Big/ \Sigma p_{i_1}^2 \cdots p_{i_c}^2 \, p_{i_{c+1}} \cdots p_{i_m}$$

where summation extends over all parmutations of m distinct integers selected from (1,2,..., n).

REFERENCES

1. Aven O.I., Boguslavsky L.B. and Kogan Ya.A. Some Results on Distribution-Free Analysis of Pagaing Algorithms, IEEE Trans. Comput., vol. C-25, No.7, pp. 737-745, July 1976.

2. Smith A.J. A Modified Working Set Paging Algorithm,, IEEE Trans.Comput., vol. C-25, No.9, pp.907-914, September 1976.

Lapozási algoritmusok egy hierarchikus osztálya

Kogan J. A.

A dolgozat a lapozási algoritmusoknak egy olyan tág osztályát vizsgálja, mely tartalmazza a legtöbb ismert tipikus stratégiát.

Р Е З Ю М Е

Об одном классе алгорифмов виртуальной памяти

Я.А. Коган

В работе рассматривается широкий класс алгорифмов, включающий большинство известных типичных стратегий.

Modeling and Simulation of Computer Systems with Simulation Nets

by

Franz Schumacher

Gesellschaft für Kernforschung mbH, Karlsruhe
D-7500 Karlsruhe, Postfach 3640, FRG

1. Introduction

The growing complexity of modern computer systems has made performance evaluation results more and more difficult to obtain. One popular approach that has been used for evaluating proposed computer systems is discrete event simulation.

An efficient and generally known model description method does not exist in the simulation of discrete systems. Especially when simulating with SIMULA or SIMSCRIPT a complete description of the simulation model will be the program itself.

The purpose of the paper is to introduce a method of representation that is useful in constructing and evaluating a simulation model in a relatively small amount of time. The method is based on Petri nets /1, 2/ and carries on the activities on Petri nets performed by Nutt /3/. The method will be named simulation net, abbreviated s-net.

The s-net system is a computer program for simulating s-nets. The s-net system has been implemented as a prefix class in SIMULA.

In the following an informal definition of an s-net is given. The application of the modeling method and the simulation system to a queuing model of a computer system is shown.

2. Definition of a Simulation Net

The basic structure of a simulation net is a directed and coherent graph with two disjoint node sets: the set of places, represented as circles, and the set of transitions, represented as line segments. Each directed arc, represented as an arrow, connects one place with one transition or vice versa, i.e. connections between places or between transitions are forbidden. Figure 1 shows an example of a net structure.

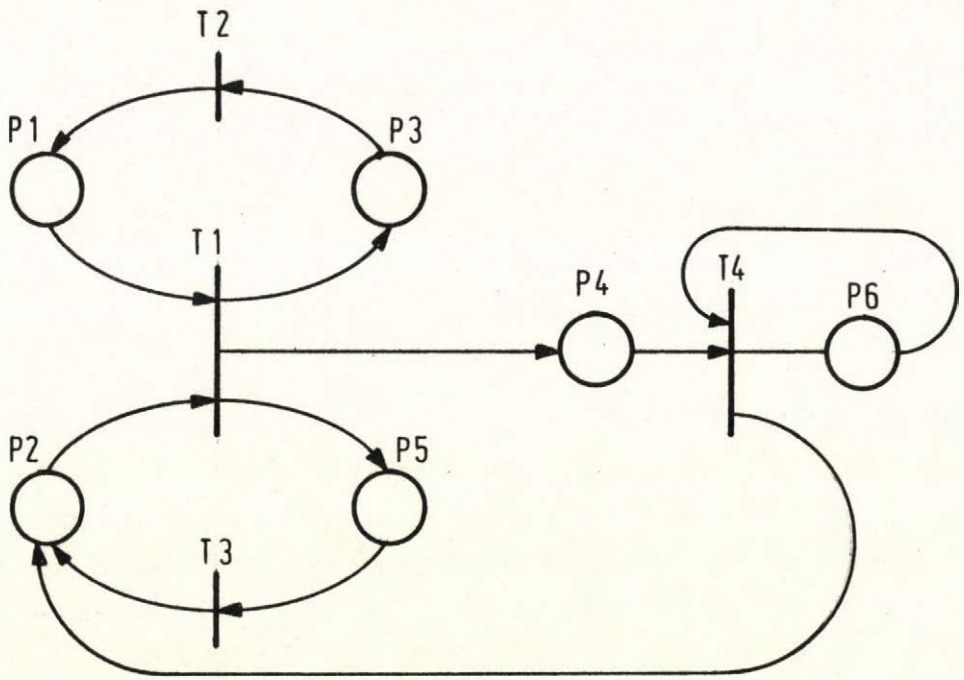


Figure 1. Example of an s-net graph

An arrow from a place to a transition means that the place is an input place of the transition, e.g. P1 of T1 in Figure 1; an arrow from a transition to a place means that the place is an output place of the transition, e.g. P1 of T2 in Figure 1.

A place may be marked, i.e. a place may contain tokens, represented as dots. For example, places P1 and P6 in Figure 2 contain one token each, and place P2 contains two tokens. Thus, the structure of an s-net is equal to the structure of a Petri net.

Every transition possesses a so-called activation scheme and transition scheme. The activation scheme of a transition contains an integer number for every input place. A transition is called activated, if the number of tokens on each input place is greater or equal to the corresponding integer number in the activation scheme. For instance, if the activation scheme of transition T1 in Figure 2 is defined by

input place no.	P1	P2
number of tokens	1	2

then the transition T1 in Figure 2 is activated.

The definition of the transition scheme is quite lengthy and will be given later. Now a transition scheme shall be defined by removing one token from each input place of an activated transition and by setting one token on each output place. This definition is the transition rule in Petri nets. The execution of the transition scheme is called firing of the transition. E.g. the transition T1 in Figure 2 is activated with the activation scheme defined above and the transition scheme can be performed. Figure 3 shows the net after firing of T1. If all the activation schemes of transitions T2, T3 and T4 contain the number one, these transitions can be fired (s. Fig. 4).

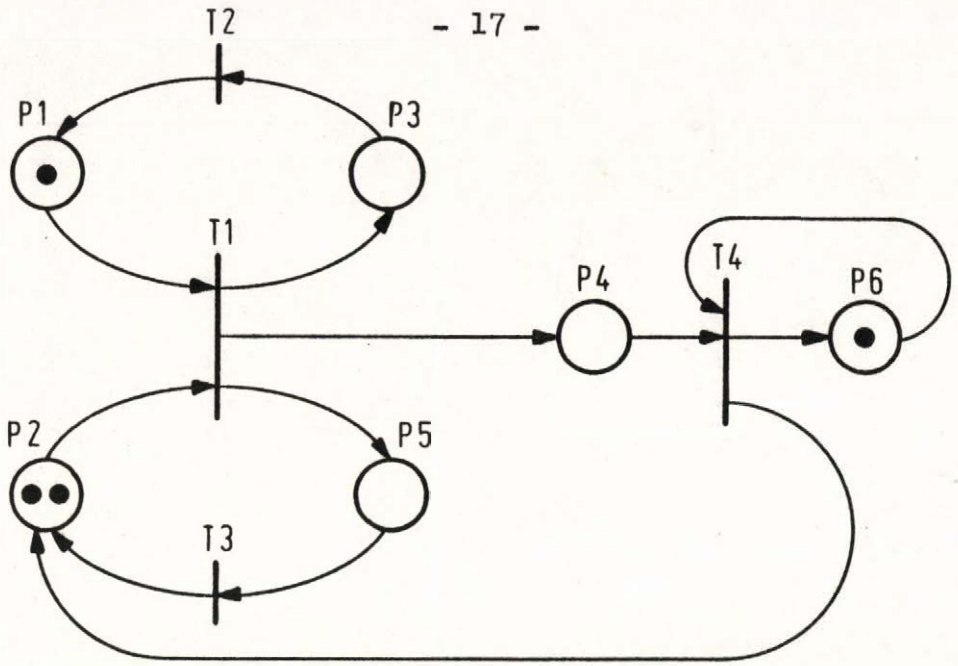


Figure 2. Example of a marked s-net graph

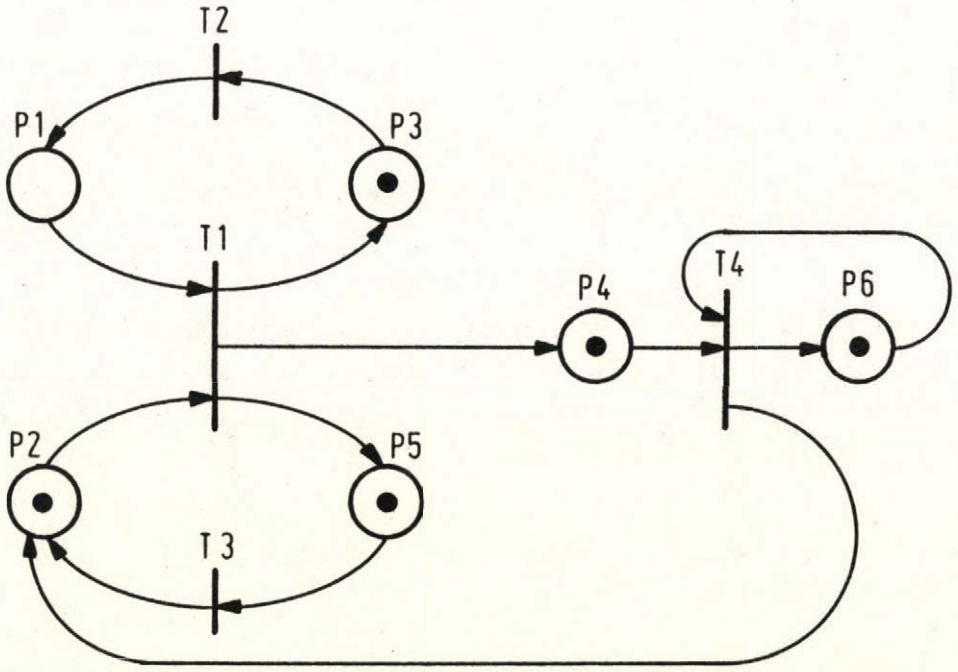


Figure 3. Fig. 2 after firing of transition T1

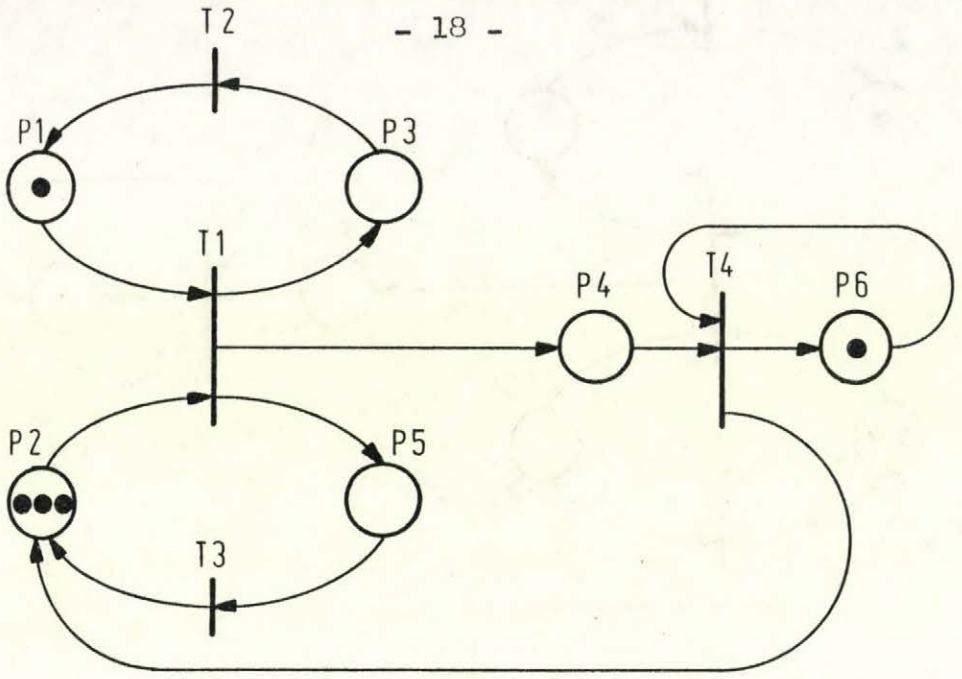
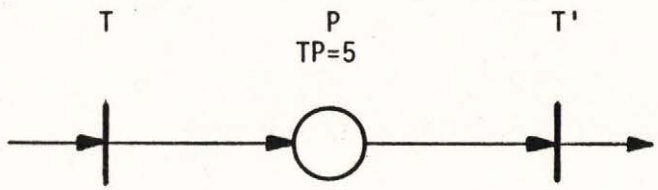


Figure 4. Fig. 3 after firing of transitions T2, T3, and T4

When s-nets are used to model and simulate discrete systems, places, transitions, and tokens may be interpreted as follows: A place represents one or more conditions. If a place is marked, these condition holds. A transition represents an event and firing of a transition means that this event takes place. A token may represent a temporary element and a place a permanent element of a system; e.g. a token may represent a job in a computer and a place may be interpreted as a CPU. If the place is marked, the CPU is busy, etc.

Every token and every place may possess parameters, e.g. token number, place number, etc.

A so-called place time TP is allocated to a place. If a token enters a place, it is delayed by this place by TP time units. E.g. if the place time of a place P is 5 time units and if a token enters place P after firing of T, then T' can't be fired before 5 time units:



The main rule of the dynamic behavior of an s-net says: If a token enters a place P with place time TP, the activation schemes of the output transitions of place P, i.e. the transitions for which place P is the input place, are called after TP time units. If the activation scheme of an output transition yields the value true, i.e. the number of tokens on each input place is greater than or equal to the corresponding integer number in the activation scheme, the activation scheme calls the transition scheme, i.e. the transition fires. If a place is the input place of more than one transition (conflict case), the activation schemes of the transitions are called in the sequence of increasing transition number.

The definition of the dynamic behavior allows the modeling of interrupts and logical structures. The place time of place P2 in Figure 5 is interrupted by place P1, because transition T' is activated and fired after finishing of place time TP1.

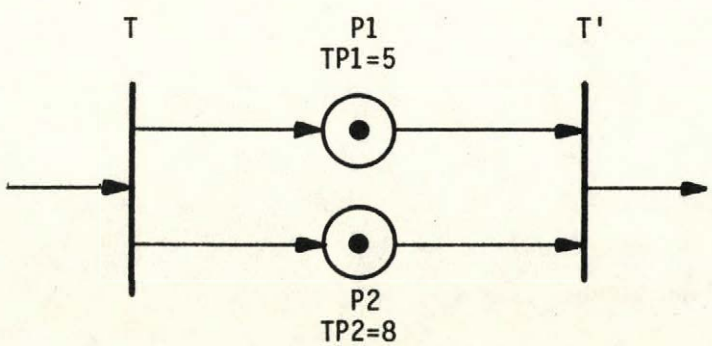


Figure 5. Example of an interrupt structure

Figure 6 shows the modeling of an inclusive - or logic. According to the resolution of the conflict case above, the transitions T1, T2, and T3 are checked in increasing transition number order.

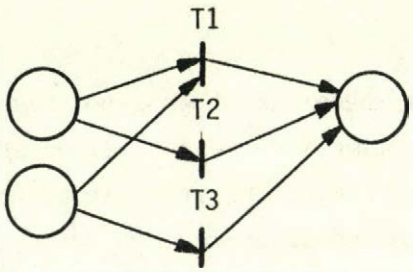


Figure 6. Modeling of inclusive - or logic

Now the transition scheme shall be defined. The scheme consists of three parts; first, the attachment of the output places to the input places, and second, for each attachment the amount of tokens to be fired are defined. The third part contains assignment statements for changing parameter values of the tokens and the places connected to the transition. For example, the table below shows a possible transition scheme of transition T1 in figure 2:

input place no.	1	2	0
output place no.	3	5	4
amount of tokens to remove	1	2	1
amount of tokens to add	1	1	1

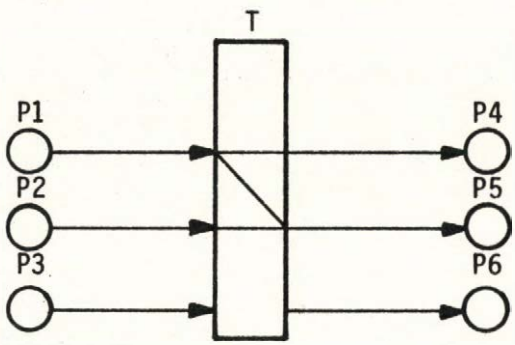
```
Place(1).Par1 = Place(2).Par1 + Token(Place(1)).Par2
```

Each column represents one link between an input place and an output place. It should be noted that in case of sinks or sources the output or input place number is zero, e.g. column 3.

The first two rows define the link.

The third row determines the number of tokens to be removed from the input places and the fourth row determines the number of tokens which are added to the output places. Under the table an assignment statement is shown which changes a parameter of input place P1.

Two special cases give rise to separate treatment, first, if one input place is connected to more than one output place, and second, if an input place contains more tokens than tokens should be fired. In these cases a selection must be made, e.g. random, first-in-first-out (FIFO), etc.. Figure 7 shows an example of a complete transition definition.



Activation scheme:

input place no.	1	2	3
minimum of tokens	1	2	4

Transition scheme:

input place no.	1	2	3	0
lower output place no.	4	5	0	6
upper output place no.	5	5	0	6
amount of tokens to remove	1	1	4	0
amount of tokens to add	1	3	0	2
token selection mode	LIFO	FIFO	FIFO	
place selection mode	RANDOM			

Figure 7. Example of a transition definition

3. The S-Net System

The s-net has been implemented as a prefix class in SIMULA. The prefix class, called SIMULATION NET, consists of the following parts:

- s-net simulation part
- selection procedures: place time - token - place
- measurement procedures
- test of the model
- input of model definition
- output procedures:
  model - results - histogram - trace - error - warning
- trace procedures
- reset procedures
- start simulation
- main line

For a more detailed description of the system see /4, 5/.

Now, the user has only to define his s-net model on input data, i.e. matrices which define the net structure, and then the simulation goes on. The standard output yields characteristics for every place relating to the maximum, minimum, average, variance of number and duration of tokens on a place, i.e. queue length, waiting times, busy period, etc.. Special routines have been implemented which support the user of the s-net in testing, data analysis etc., e.g. trace, histogram, reset and clear procedures.

4. An Example

A model of a computer with three parallel CPU's and two IO-devices is shown as queuing model in Figure 8. The queuing disciplines before the IO-devices and the CPU's are assumed to be FIFO. The selection mode of an IO-device shall be the shortest queue length rule. The service times in the IO-devices are Erlang distributed with parameters 0.2, 3 and 0.2, 2, respectively. The CPU time is exponential distributed with parameter 0.1 . 50 jobs are in the system.

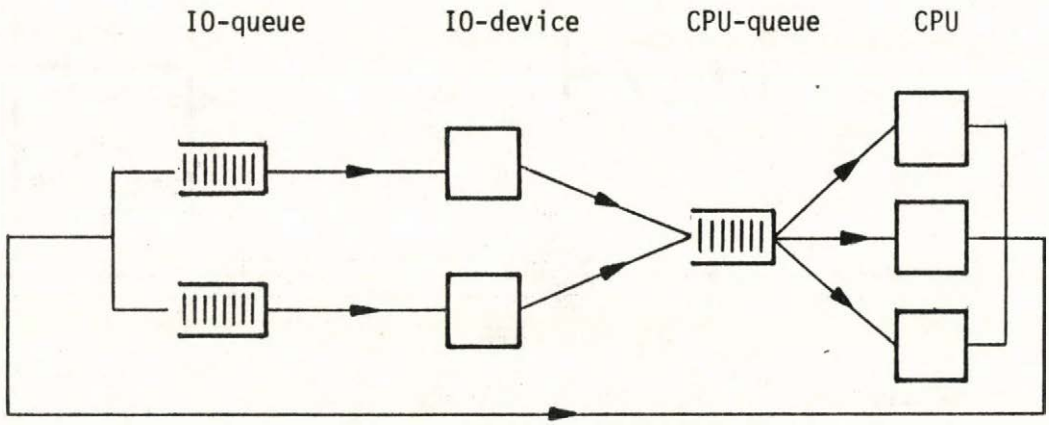
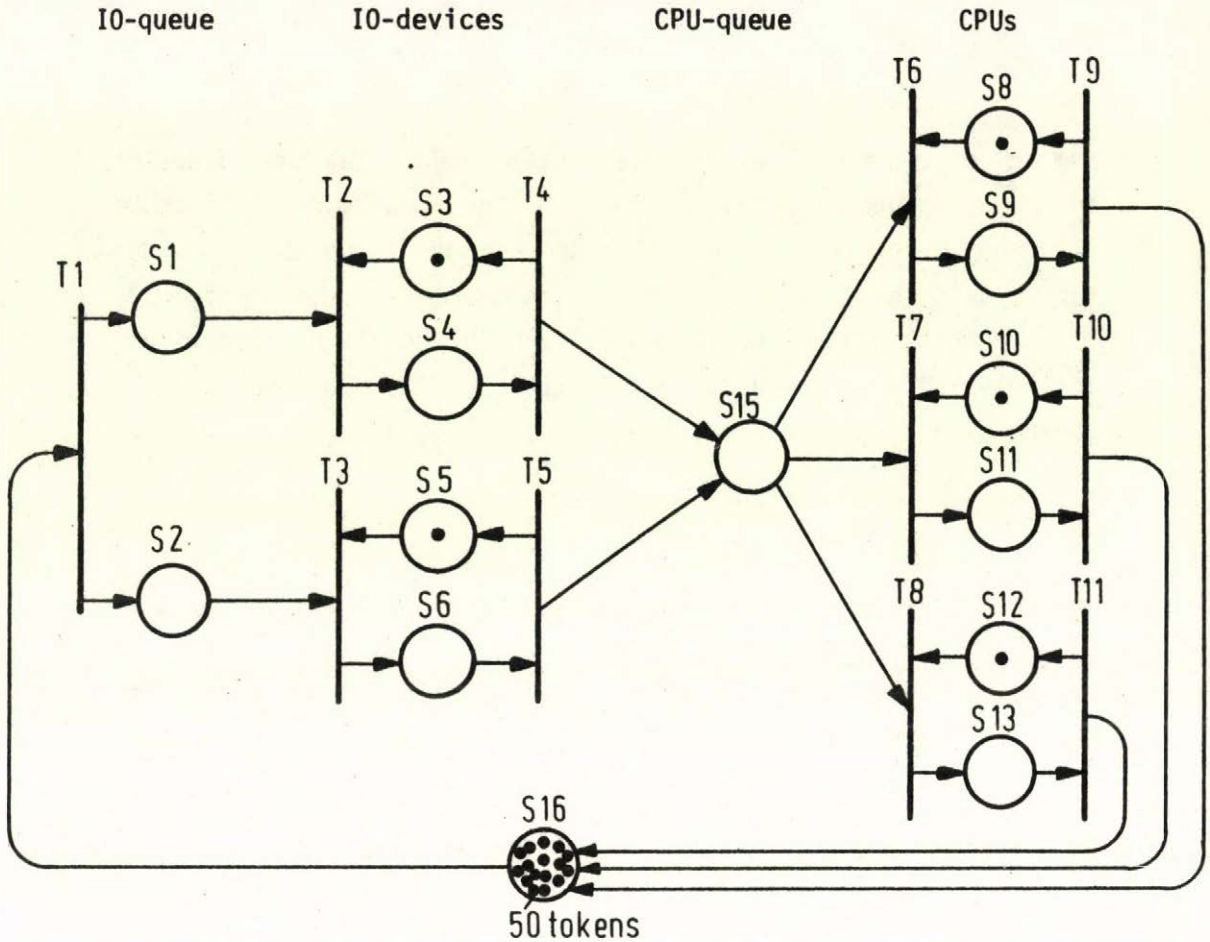


Figure 8. Cyclic queuing model of a computer

The structure of the s-net model is shown in Figure 9.

The output of the s-net system for the model is shown in the appendix.

The first table in the output report contains the random sample amount of the following tables. The dwelltime can be interpreted as



place times: TP1, 2, 3, 5, 8, 10, 12, 15, 16 = 0
TP4 = erlang (0.2, 3)
TP6 = erlang (0.2, 2)
TP9, 11, 13 = negexp (0.1)

activation scheme: one token for each input place

transition scheme:

	transitions T2-11	T1
input place no.	...	16
lower output place no.	...	1
upper output place no.	...	2
amount of tokens to remove	1	1
amount of tokens to add	1	1
token selection mode	FIFO	FIFO
place selection mode		QUEUE

Figure 9. s-net of the cyclic queuing computer model

waiting time (place 1), idle time (place 3) or service time (place 13). The values in the table 'statistics of tokens' represent e.g. queue length (place 1) or throughput (place 6).

The computation time ot the example was about 30 sec CPU-time on IBM 370/168. Thus, the average computation time of one transition firing, i.e. one event, was about 2 msec.

3. Conclusions

S-nets can generally be applied to modeling and simulation of any discrete system. The advantages are manifold:

- Graphical arrangement, which means clear representation of system elements and constraints.
- Laborious programming work can be avoided almost completely and thus the verification of a program as well.
- The model definition in terms of an s-net is comprehensive and can serve as a platform for discussions with involved persons, e.g. managers, team members, etc..
- The s-nets are easy to learn, because they can be defined on few sheets of paper (no voluminous handbook).
- Simulation studies can be performed by people who are not familiar with programing languages.

The major disadvantage lies in the difficulty to survey s-net models of very complex systems.

Future possible tasks are the implementation of a display interface (light pen), the connection with description methods used in continuous simulation (e.g. DYNAMO), and improvements of the s-net

itself, e.g. representation of a model at different levels of detail (topdown approach) in order to eliminate the disadvantage mentioned above.

Acknowledgement

I want to thank Mr. R. Ramöller for implementing the s-net in SIMULA and for many discussions.

/1/ C.A. Petri — Kommunikation mit Automaten. Dissertation, Universität Bonn, 1962.

/2/ A.W. Holt et al. — Information system theory project (final report). AD 676972, Princeton, 1968.

/3/ G.J. Nutt — Evaluation nets for computer system performance analysis. Fall Joint Computer Conference, 1972.

/4/ F. Schumacher — The S-Net System User's Manual. Interner Arbeitsbericht, GfK-IDT-263-02, Karlsruhe, 1976.

/5/ R. Ramoeller — Implementierung eines erweiterten Petrinetzmodells zur Simulation diskreter Systeme. Diplomarbeit, Universität Karlsruhe, 1976.

Appendix: Output Report of the S-Net System for the Example in Figure 9.

```
R E S U L T S   O F   M E A S U R E M E N T:
**********************************************

 ABSOLUTE TIME:      11000.0000
 SIMULATIONTIME :    10000.0000

 STATISTIC OF PLACES
 ***********************
```

PLACENO	PLACENAME	TOTAL OUTPUT	CURRENT
1	IO-QUEUE	1788	0
2	IO-QUEUE	1195	0
3	IO-DEVICE1(IDLE)	1788	0
4	IO-DEVICE1(BUSY)	1788	1
5	IO-DEVICE2(IDLE)	1195	0
6	IO-DEVICE2(BUSY)	1195	1
8	CPU 1-2-3 (IDLE)	956	0
9	CPU 1-2-3 (BUSY)	956	1
10	CPU 1-2-3 (IDLE)	1002	0
11	CPU 1-2-3 (BUSY)	1002	1
12	CPU 1-2-3 (IDLE)	1024	0
13	CPU 1-2-3 (BUSY)	1024	1
15	CPU-QUEUE	2982	45
16	CPU-IO CYCLE	2982	0

```
 STATISTIC OF DWELLTIME
 **************************
```

PLACENO	PLACENAME	AVERAGE	STAND.DEVIATION	MAXIMUM	MINIMUM
1	IO-QUEUE	5.604	5.862	33.186	0.000
2	IO-QUEUE	5.415	6.484	39.288	0.000
3	IO-DEVICE1(IDLE)	0.688	2.094	19.246	0.000
4	IO-DEVICE1(BUSY)	4.909	2.806	16.611	0.187
5	IO-DEVICE2(IDLE)	3.211	8.061	77.695	0.000
6	IO-DEVICE2(BUSY)	5.160	3.588	28.746	0.026
8	CPU 1-2-3 (IDLE)	0.000	0.000	0.000	0.000
9	CPU 1-2-3 (BUSY)	10.449	10.636	78.312	0.004
10	CPU 1-2-3 (IDLE)	0.000	0.000	0.000	0.000
11	CPU 1-2-3 (BUSY)	9.966	10.217	73.291	0.039
12	CPU 1-2-3 (IDLE)	0.000	0.000	0.000	0.000
13	CPU 1-2-3 (BUSY)	9.793	9.520	54.111	0.008
15	CPU-QUEUE	147.257	22.578	219.500	95.445
16	CPU-IO CYCLE	0.000	0.000	0.000	0.000

```
 STATISTIC OF EMPTYTIME
 **************************
```

PLACENO	PLACENAME	AVERAGE	STAND.DEVIATION	MAXIMUM	MINIMUM
1	IO-QUEUE	0.397	1.603	24.367	0.004
2	IO-QUEUE	0.612	3.627	79.941	0.014
3	IO-DEVICE1(IDLE)	0.878	2.223	16.611	0.188
4	IO-DEVICE1(BUSY)	0.123	0.923	19.246	0.000
5	IO-DEVICE2(IDLE)	0.616	2.083	28.746	0.026

6	IO-DEVICE2(BUSY)	0.384	2.974	77.695	0.000
8	CPU 1-2-3 (IDLE)	0.999	4.499	78.312	0.004
9	CPU 1-2-3 (BUSY)	0.000	0.000	0.000	0.000
10	CPU 1-2-3 (IDLE)	0.997	4.400	73.291	0.039
11	CPU 1-2-3 (BUSY)	0.000	0.000	0.000	0.000
12	CPU 1-2-3 (IDLE)	1.003	4.251	54.111	0.008
13	CPU 1-2-3 (BUSY)	0.000	0.000	0.000	0.000
15	CPU-QUEUE	0.000	0.000	0.000	0.000
16	CPU-IO CYCLE	1.000	2.441	28.004	0.004

STATISTIC OF TOKENS
****

PLACENO	PLACENAME	AVERAGE	STAND.DEVIATION	MAXIMUM	MINIMUM
1	IO-QUEUE	1.001	1.081	6.000	0.000
2	IO-QUEUE	0.647	0.979	6.000	0.000
3	IO-DEVICE1(IDLE)	0.123	0.328	1.000	0.000
4	IO-DEVICE1(BUSY)	0.877	0.328	1.000	0.000
5	IO-DEVICE2(IDLE)	0.384	0.486	1.000	0.000
6	IO-DEVICE2(BUSY)	0.616	0.486	1.000	0.000
8	CPU 1-2-3 (IDLE)	0.000	0.000	1.000	0.000
9	CPU 1-2-3 (BUSY)	1.000	0.001	1.000	0.000
10	CPU 1-2-3 (IDLE)	0.000	0.000	1.000	0.000
11	CPU 1-2-3 (BUSY)	1.000	0.001	1.000	0.000
12	CPU 1-2-3 (IDLE)	0.000	0.000	1.000	0.000
13	CPU 1-2-3 (BUSY)	1.000	0.001	1.000	0.000
15	CPU-QUEUE	43.844	2.437	47.000	33.000
16	CPU-IO CYCLE	0.000	0.001	2.000	0.000

PATHS
****

FROM PLACE	UNTIL PLACE	AVERAGE	STAND.DEVIATION	MAXIMUM	MINIMUM
16	15	10.535	6.812	43.350	0.304

PATHS
****

FROM PLACE	UNTIL PLACE	AVERAGE	STAND.DEVIATION	MAXIMUM	MINIMUM
16	16	167.918	25.525	261.309	104.309

HISTOGRAM : FOR PLACE NO. 15

DWELLTIME

	FREQUENCY	
0.0E+00	17	******
0.00E+00	8	***
5.00E+00	2	*
1.00E+01	1	
1.50E+01	1	
2.00E+01	0	
2.50E+01	6	**
3.00E+01	10	***
3.50E+01	1	
4.00E+01	7	**
4.50E+01	5	**
5.00E+01	13	*****
5.50E+01	21	*******
6.00E+01	19	*******
6.50E+01	0	
7.00E+01	4	*
7.50E+01	11	****
8.00E+01	6	**
8.50E+01	14	*****
9.00E+01	20	*******
9.50E+01	23	********
1.00E+02	81	*****************************
1.05E+02	107	**************************************
1.10E+02	133	**
1.15E+02	161	***
1.20E+02	232	**
1.25E+02	214	**
1.30E+02	242	**
1.35E+02	266	**
1.40E+02	288	**
1.45E+02	273	**
1.50E+02	235	**
1.55E+02	241	**
1.60E+02	162	***
1.65E+02	111	**************************************
1.70E+02	101	**********************************
1.75E+02	54	******************
1.80E+02	57	*******************
1.85E+02	41	**************
1.90E+02	48	****************
1.95E+02	46	***************

Számitógéprendszerek modellezése szimulációs hálókkal

F. Schumacher

A dolgozat egy a Petri-hálókon alapuló módszert ismertet számitógéprendszerek müködésének leirására, ismerteti annak megvalósitását SIMULA osztályként, és bemutatja módszerének alkalmazását.

Моделирование вычислительных систем с помощью имитационных сьтей

Ф. Шумахер

Работа знакомит читателя с некоторым методом основанным на сетях Петри для описания работы вычислительных систем. В статье описывается текже осуществление моделирования, как класс и р прилагаются применения метода.

OSCAR
Operating System for Computer Aided Design's Realization.

J. Fabók, G.Hermann, Zs.Rácz
Computer and Automation Institute
of Hungarian Academy of Sciences

INTRODUCTION

The paper describes an automated workshop of special purpose machines controlled by a minicomputer. This workshop is dedicated to design, produce and test printed circuit cards and electrical equipments.
The information flows in the system in two ways. The description of the object to be produced can be given in a pretty formal way to a midicomputer - in this case a CDC 3300 - where it is processed by a special program package and forwarded either by a communication line or by hand through a punched tape. The other way is to produce these data in a very primitive language by the designer and give it directly to the minicomputer. Therefore there must be a possibility to check those descriptions for correctness, modify it if necessary, check it again and so on in an interactive way.

The information, produced in either way and stored in the library, is considered a data set from the aspect of the OS, but for the special purpose machine it is a program, prescribing its action. This paper deals with the OS, so the library files will be referred to as data.

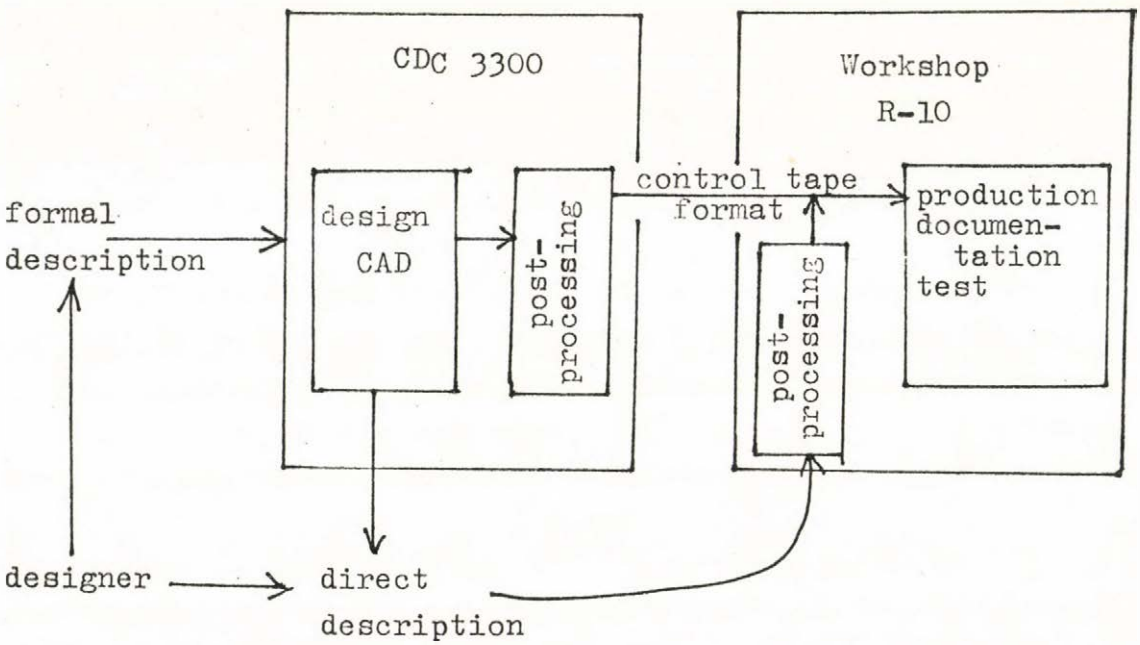
The operation of the workshop is controlled by an operator sitting before a keyboard of a teletype or a display. It is her duty to initialize all works to be done concerning both the production and the interactive design process. This is performed by programs and by the OS whose action is pretty predetermined.

Taking into account the requirements to be met by a so-called general purpose OS and by this one, you may conclude that this special OS, providing these facilities, is an extended subset of the "general purpose" OS.

We decided to build up taylor made system, i.e. some well known facilities had been left out /e.g. the possibility of extension or modification of computer programs/, and on the other hand new ones corresponding to the actions to be performed were introduced.

INTEGRATED SYSTEM

The integrated system developed in our Institute is a complex of hardware and software and its scheme is outlined in fig.1.



Integrated system

fig.1.

The design is always the duty of the CDC 3300 and the output which is a technological description of the object in the alphanumerical direct description language must be processed by postprocessors to get the data called control tape format digestable for the special purpose machine. Postprocessing may take place in either computer. The last phase of the process, i.e. production, documentation and test takes place in an automated workshop. In the following, in order to distinguish the electrical ewuipment to be designed and produced from the equipment operating in the workshop, the former group will be referred to as "equipment" and the latter as "dedicated machines" or simply "machines". Using the control tape as an input, the machines are able to work off-line, too. The work in the workshop takes place automatically, controlled by a minicomputer R-10 made in Hungary. On-line, the machines can operate on the basis of both the numeric control tape format and the direct description, called in the following symply data files.

FUNCTIONS OF THE WORKSHOP COMPUTER

The tasks of the workshop computer are the following:
1. Establishing information flow between the computer and the machines, which means to operate the machines and collect information from them.
Operating the machines supposes a two way connection between the computer and the machines. The computer sends the dedicated machine control bytes or normal data bytes. A control byte is an instruction to the machine itself, /for instance: start, stop, send status message, etc./ Normal data bytes are parts of the numeric control tape format. If the production takes place on the basis of the direct technological description , the conversion to the control tape format is performed real-time.

A machine sends the computer data bytes or status messages. The latter consists of a status byte and supplementary message if one byte is not able to express all the information to be sent.
There is a possibility of the data collection for mainly statistical purposes.
2. Providing possibilities to modify the description of the equipment to be produced. This may be necessary mainly in the case when the technological description is given directly to the system by the designer, so it is not produced by the midi computer. The picture of the cards can be plotted and the designer engineer has the possibility to check and correct them in the workshop using these facilities.
Only the correct cards will be manufactured. Even in the case of an automatic design, the modification of data files may be demanded. First of all, the automatic design handles only certain elements of the cards. To use other elements, for instance, condenser, etc., one has to modify the designed card by putting these elements on it.
The machines in the workshop are operated in parallel, so multiprogramming capability is a very important feature of the system.

HARDWARE ENVIRONMENT of the workshop.

The dedicated machines in the workshop are as follows.

ADMAP	it produces printed circuit card plates
CALCOMP	it is a special plotter
MSI TESTER	it carries out the test of IC-s
TESTOMAT-C	it tests ready printed circuits
MANU WRAP	and TSK are semiautomatic machines for wire-wrapping.

Most of these machines have been made in our Institute, but there are also foreing made machines among them.

These machines operate without any disturbances even if the speed of the data flow decreases from time to time. So the time is not a crucial factor in their handling. The connection has been realized by BSI /British Standard Interface/.

The system peripheral equipment are: one disc, one paper tape unit and two teletypes, one to control the workshop, one for editing. At most 6 dedicated machines can be connected to the system.

SOFTWARE COMPONENTS

The whole software consists of the following components:

special operating system,
special jobs,
data files.

In the sequel the special OS will be discussed in detail. Now the question arises, why such a structure of software is useful in solving the tasks mentioned above.
The individual machines of the workshop require various and different services from the system. These differences were so big that it didn't pay off to furnish these services by one single program. Instead, the system contains a set of programs, called functional programs, each devoted to one specified function of a specified machine. These functional programs form a significant part of the jobs, processed in the system.

An other kind of jobs are the service programs. They make it possible to modify a data file written in an alphanumerical language /editors/, to analyse it with respect of syntax /syntax analyzers/, to convert it into a numerical control tape format /postprocessors/, and to plot the picture of the object in order to check it before production. Since the direct technological description of the object depends on the machine it is made for, so as many postprocessors and syntax analyzers

are needed, as many types of the dedicated machines exist. These service programs are not built into the system constantly, they can be changed in case of necessity.

These jobs are executed in parallel in the system. They are monitored by the special OS i.e. they are put in, initiated and finished by the OS. In addition to this, the OS takes care of resources shared by the processes, of data management, of standard I/O and of some other common functions.

We want to emphasize that the OS supervises only the execution of the jobs. In the system only the data files can be modified. To create, to debug and to modify both functional programs and service programs are possible only by the host computer, and this is valid for the parts of the OS, too.

FUNCTIONAL PROGRAMS

A functional program describes the function the computer has to execute to operate a dedicated machine. It usually begins with testing the status of the machine and specifying the data file and its record to be worked upon. If the machine is ready to work, the functional program begins to send data bytes to the dedicated machine. It receives and analyzes the status messages from the machine and according to these messages it decides about the further operation of the machine.

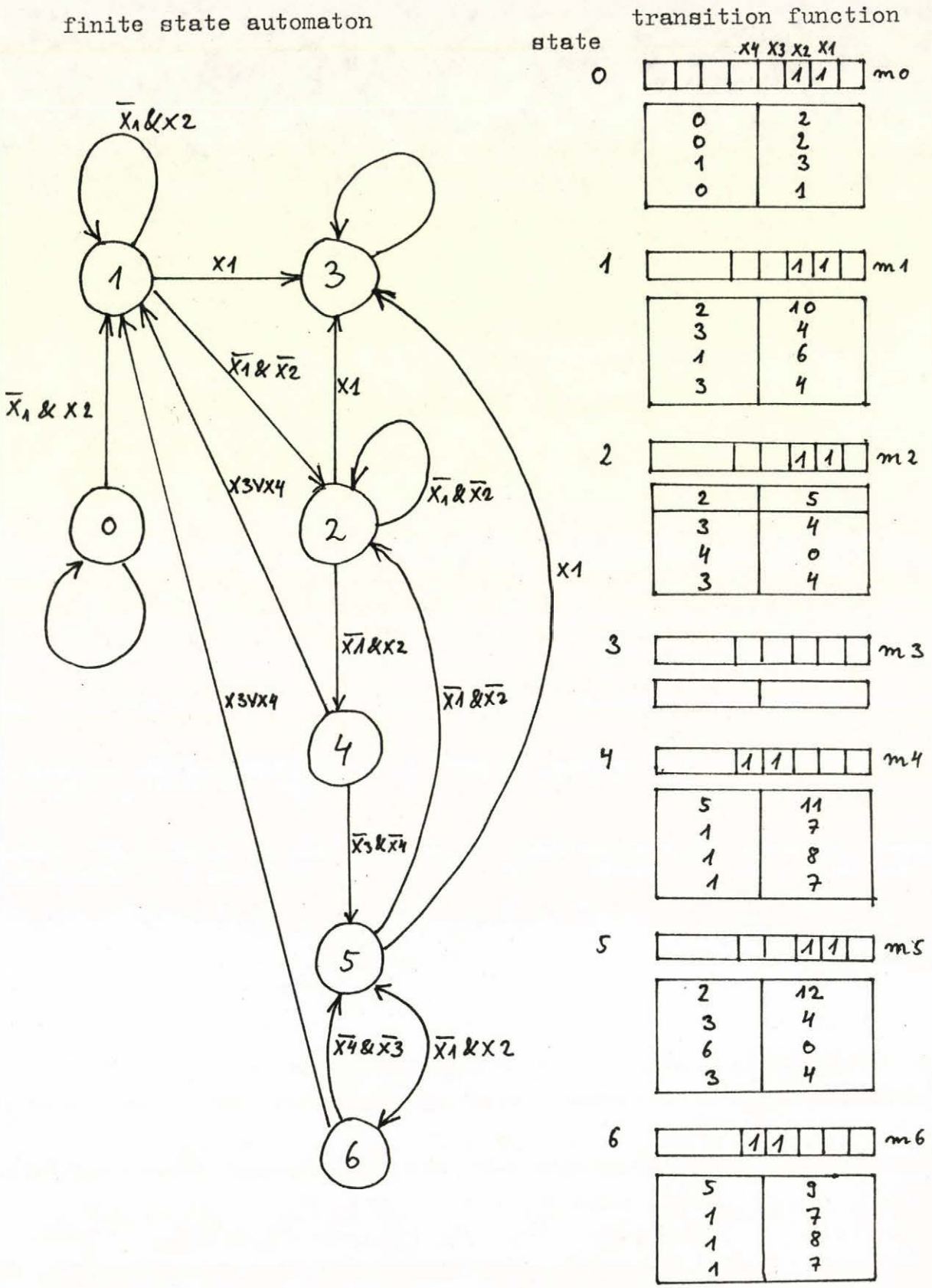
A functional program consists of two main parts, the operational program and the emergency program. The operational program takes care of testing the machine, specifying the data set, sending data bytes or blocks to the machine and communicating with the operator. The emergency program is responsible for the analysis of status messages. If there is a change in the status of the machine, it sends a status

message informing the system about its new status. An arrival of a status message stops the execution of the operational program, the status message is analysed by the emergency program, and a decision is made about the further action.
The result of this decision may be:

- to continue the operational program without any modification
- to continue the operational program at a new entry point
- to suspend all operations until a new status message arrives.

The emergency program is executed on a higher interrupt level than the operational program. Both the operational program and the emergency program use the facilities of the OS.

The analysis of the status messages are standardized as follows. The state of a machine is described in the computer by one or more finite state automata. The handling of some machine actions, mainly in connection with the start and the stop, is common for all machines. These actions are described by a common automaton. The individual actions, characteristic only for a certain machine, are described by its private automaton. Transition functions are represented in form of tables. The input of the automaton is determined by the status message as well as some logical variables evaluated during the execution. The output is a series of actions, which is referred to as the entry point of the program describing these actions. Evaluation of the state transition is made by reentrant system routine determining the new state and the output of the automaton according to the current state, the transition function and the input signal.
The common automaton and its transition funcition can be seen in fig. 2. The following bits of the input signal are handled by this automaton:



Finite state automaton and its transition function.

fig. 2.

x1 - on-line: 0, off-line: 1
x2 - start: 0, stop:1
x3 - anew: 1, else: 0
x4 - consol request:1, else: 0

The bit x3 is used to indicate the request of returning to the beginning of the actual record.
The bit x4 shows the operator's request to interfere with the work of the machine. The zero value of xi will be denoted as $\overline{xi}$. An arrow without any inscription marks all the rest possibilities.
Masks are used to decrease the size of the transition function tables. The mask mi shows the bits of the input signal significant in the state i.
The output signal 0 is a special one. It means that the same input signal will be evaluated in the new state of the automaton. This is another tool to decrease the size of the table.

FUNCTIONS AND SPECIALITIES OF OSCAR

Its functions are:

- to give possibility to establish a new job,
- to provide a multiprogramming capability and to supervise the common resources,
- to give standard data management,
- to provide I/O facilities,
- to organize the operator communication,
- some other special functions, for instance to evaluate the state of automata, etc.

Since the structure of usual operating systems is well known, only the special features of this OS will be discussed here.

The following resources are shared in OSCAR:

- CPU

- operative memory
- mass-storage /disc-files/
- system peripheral equipments
- standard routines of the OS.

In connection with the memory allocation, it is an important fact, that during their execution the functional programs are in the operative memory apart from the overlay structure defined by the programmer of the functional program. As a result the memory allocation is in close connection with the initiation and termination of the jobs.

Another very important fact is that the dedicated machines are not shared among the jobs. From this point of view the peripheral equipments including the dedicated machines can be devided into 3 groups.

a./ shared devices. Processes waiting for such a device form a queue. A device is only temporarily allocated to an actual process. When the required operation has been completed, the process releases the device.

b./ reserved devices. In this case, it is not worth waiting for such a device, because the reservation may take long time, for instance hours. Reservation is marked by a flag. If the called device is busy, the calling process will be aborted issuing an error massage. The dedicated machines belong to this group.

c./Exclusively system equipments. These devices can be used only by the OS and not by the jobs. So there isn't any reservation.

The standard routines of the OS are reentrant. They operate the data field of the calling process.

The different services of the OS may be used by operator communication through the consol or by calling standard routines from the functional programs.

Standard routines include the followings:

- I/O actions /transfer of bytes or blocks from and to the dedicated machine/
- routines building a part of the data management /to get bytes, blocks or pages of data files/
- routines for operator communication /consol handler/
- routines called by emergency programs

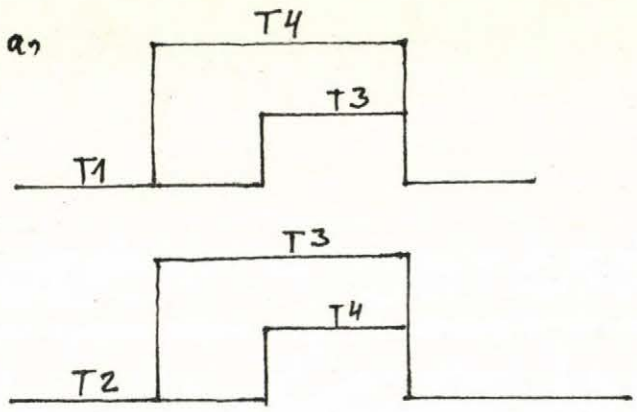
The following functions are initialised by operator instructions:

- job management /to establish, to initiate and to terminate jobs/
- a part of data management /to put in and delete files, etc./
- some other functions /to list jobs and machines, to punch data files, etc./

In connection with the resource allocation, the deadlock situations must be discussed. A usual deadlock situation can be seen in fig. 3/a. T1, T2, T3 and T4 are tasks, T3 and T4 are handlers of some resources. T1 is waiting for T3 and T2 is waiting for T4, but they can't get the required resources because the resources are reserved by the other task. But in our case, as we mentioned before, it is not allowed for a task to call two tasks in parallel, so this deadlock situation can't occur in the system.

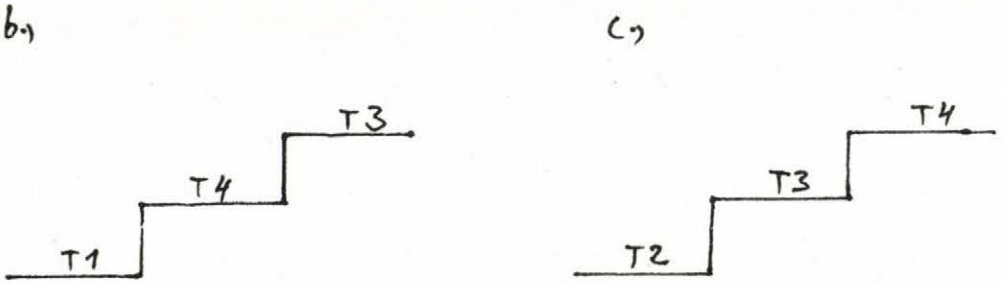
Figure 3/b and 3/c show a deadlock situation between system tasks. This case may actually accur, for instance T4 is a line printer handler and T3 is a disc handler, so in fig. 3/b the task T1 calls the line printer handler, and the line printer handler calls the disc handler because the text to be printed is stored on the disc. In fig. 3/c the disc handler calls the line printer handler to print a disc error message. This situation results a deadlock. To avoid this situation,

a priority order must be determined among the system tasks in respect of calling. For instance T4 may call T3, but T3 musn't call T4, i.e. we have forbidden that a disc error massage be printed on the line printer.
We use other method to indicate a disc error. So the situation in fig. 3/b is allowed and the situation in fig. 3/c is forbidden.



T1, T2, T3, T4 — tasks

Tipical deadlock situation



T4 : Line printer
T3 : Disc handler

Deadlock situations
fig. 3.

LOW LEVEL SCHEDULING

Low level scheduler allocates the CPU among the processes ready to run, i.e. it makes a choice of which process to run first. To understand the scheduling strategy it may be useful to be familiar with the interrupt levels and the I/O handling in the system.

4 interrupt levels serve for all the dedicated machines.These are in the order of their priority:

- IT level of status messages of the machines /emergency level/
- IT level of input data bytes /INP level/
- IT level of output bytes, control bytes or normal data bytes /OUT level/
- IT level of operational program /MAIN level/

IT level of status messages has the highest priority. Transfer of a data block is initialized on the MAIN level, but the transfer is carried out on the according IT level. An IT on the emergency level causes to stop both the operational program and the data block transfer.
Decision of the emergency program may cause the operational program to become inactive. The whole emergency program, including reading and analyzing the status message, is executed on the emergency level. Therefore a new eventual initiation of an emergency program may take place only after the current one has been finished, so these programs need not be reentrant.

The status of a process may be the following: ready, suspended, running, dormant.Their meaning are the same as usual. A living process gets over the following state transitions:

running → suspended → ready → running.

In dormant state the process is inactive. It can get into dormant state from every state in consequence of the decision of the emergency program. Similarly only a status message can "awake" the process, i.e. bring it out from the dormant state. Processes in ready state are competing for the CPU and processes in suspended state are waiting for one of the resources. Both ready and suspended processes form queues /first in first out/.

The core of the information, used by the OS is the system table. The cells of the system table are owned by the dedicated machines. In case a process is initialised it will be associated to the cell of that dedicated machine the process is operating on, This cell contains the following information:

- queue linkage
- process status
- pointer to the data field of the process

As it is not allowed for a process to wait for two resources in parallel, so a process occurs only in one queue at the same time. The first part of the data field of a process is standard. This serves to hold information for standard reentrant routines, for instance the contents of registers, pointers to the actual data, working storage, etc.

An important characteristics of our scheduling strategy is that the interrupts don't cause scheduling. Control is transferred to the scheduler only in the case when the process has to wait for completion of some event, for instance I/O operation. Its advantage is simplicity. Its disadvantage is that a bad operational program can monopolize the CPU. In this system this disadvantage does not play role due to the following:

- The nature of the functional programs is such, that they often need some not available resource, so they will be often suspended /e. g. data transfer/.
- Dedicated machines are not real-time devices.

OPERATOR COMMUNACATION

The operator can interfere with the operation both of the OS and the functional programs via consol. Job management takes place in an interactive mode. First of all the operator puts in the functional programs, service programs and data files. In order to initiate a job the operator specifies the dedicated machine and the functional program, which will be associated to each other. The OS checks the free status of the machine, the existence of the functional program and the available free memory for the program. If these conditions are satisfied, the job will be initiated. If even after a garbage collection the memory is full, the job will be aborted with an error message. At the end of a functional program the operator has the possibility to repeat the execution of the program.

Besides this the following facilities of the system can be used by the operator via the consol.

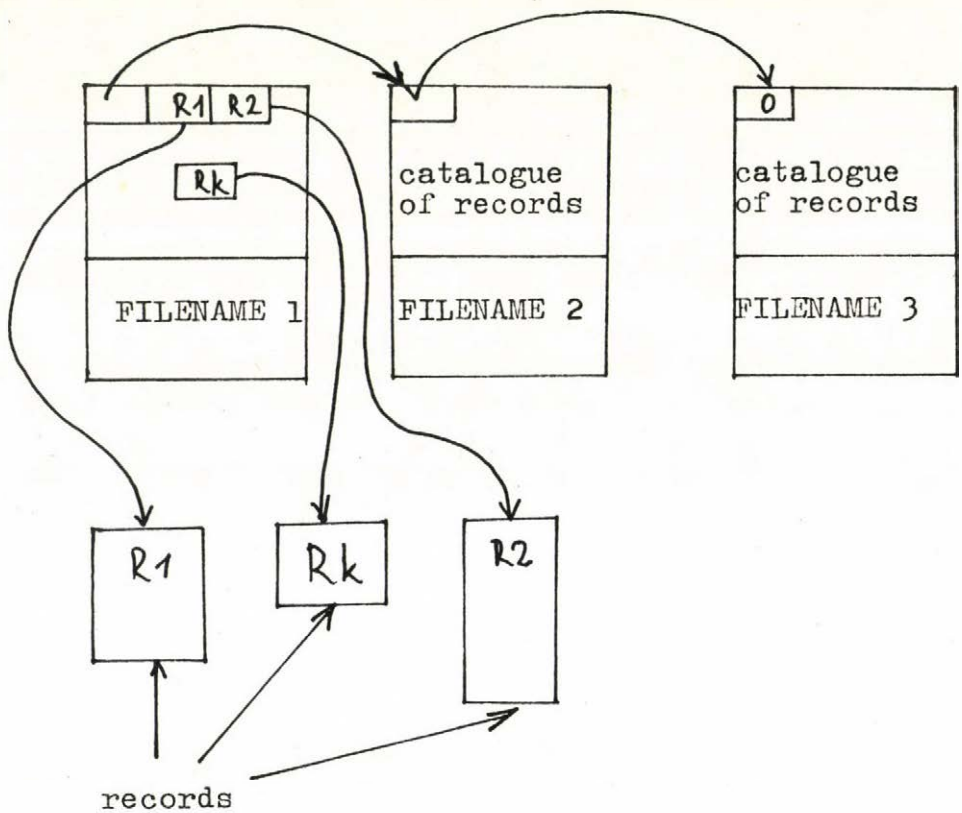
LIST - to list programs and data files
STATE - to list the state of the dedicated machines
DELETE - to delete programs and data
RENAME - to rename data
HARDCOPY - to punch data
TAPE - to transfer data to and from magnetic tape
MODIFY - to modify system tables

Functional programs can use the consol to communicate with the operator. Standard routines serve to reserve and to release the consol, to print a string of characters and to receive a character.

DATA MANAGEMENT

The structure of the data basis is very simple. Every file contains records of the same amount.

A catalogue page belongs to each file, it contains the name of the file /maximum 80 char./ and the locations of its records. The catalogue pages of the files form a linked list. Records are on consecutive pages on the disc. The location of their first pages can be found in the catalogue. The structure of the library can be seen in fig. 4.



The structure of the library
fig. 4.

Data records can be established and deleted by operator communication. Every active process has a one page puffer. In order to get data from files of the library, functional programs use standard routines. This can happen on two different levels. The program can ask the next charachter, the next block of n characters, the next block terminated by a terminator character of the current record. In this case the system takes care of the automatic change of pages. If the structure of the records is well defined, it is possible for the program to handle its data itself. This increases the efficiency of the program.

CONCLUSION

The development of our system was based on an analysis of the functions to be carried out by the OS. According to our opinion it is worth developping a special operating system to solve a special application problem, because it will be cheap and effective to operate and it is not too expensive to create. Thanks to the structure of the software, our system is extensible in the sense that new machines and new services may be easily included when demanded. The system is now operating since half a year in the RSD department of an electronic factory in Hungary.

REFERENCES

1. BACH I. FABÓK J.:
   OSCAR speciális célokra készült operációs rendszer,
   Mérés és Automatika 1975/4 126-131 Budapest, Hungary
2. BACH I. FABÓK J.:
   OSCAR operációs rendszer Számitógéptechnika '74 Esztergom

3. TSICHRITZIS, D.C. and BERNSTEIN, P.A.:
Operating Systems,
Academic Press, Princeton 1974.
4. W.F.C. PURSER and D.M. JENNINGS
The Design of a Real-Time Operating System for a Minicomputer.
Software-practice and experience Vol. 5. 147-167/1975/

Ö S S Z E F O G L A L Ó

Fabók J., Hermann G., Rácz Zs.: OSCAR

Az OSCAR egy célorientált operációs rendszer, amely elektronikus digitális berendezések gyártását, dokumentálását és ellenőrzését végző célgépek egyidejü üzemeltetésére szolgál. Ezen cikk áttekintést ad a rendszer egészéről, a hardware és software környezetről és az operációs rendszer funkcióiról. Részletesen foglalkozik a kezelő programokkal és az operációs rendszer fő komponenseivel, a task-kezelés, job-kezelés és adat-kezelés strukturájával és szolgáltatásaival.

РЕЗЮМЕ

Фабок Ю., Херманн Г., Рац Ж. : ОСКАР

ОСКАР является специальной операционной системой, служащей для эксплуатации электронного оборудования для изготовления, документации и проверки печатных плат. В статье дается общее описание всей системы, аппаратных средств и математического обеспечения, используемых данной операционной системой и ее функций. Дается подробное описание ее структуры и главных компонентов

DESIGN AND SIMULATION OF DECENTRALIZED CONTROL STRUCTURES FOR RELIABLE DISTRIBUTED SYSTEMS

B. Wolfinger, O. Drobnik, E. Holler
Kernforschungszentrum Karlsruhe
Institut für Datenverarbeitung in der Technik

1. Introduction

Computer networks generally consist of several autonomous computers (Hosts) interconnected by a communication system for message exchange between the Hosts. The application of computer networks in areas characterized by the demand for high reliability of the execution of (user) requests is increasing considerably. Reliable execution of requests can be achieved by redundancy (e.g. simultaneous execution by several computers) or the handling of a request by another computer in the case of failure of the executing computer. To be efficient and reliable, execution of requests demands the existence of coordinating mechanisms between the computers.

Most of the existing solutions to this problem apply centralized control structures - one master coordinating the work of the other computers, the slaves. This solution eventually fails if the master computer is down. The improvement of communication technology allows the development of decentralized coordination mechanisms, which are more flexible and still work correctly if some of the coordinating computers fail. A protocol for decentralized control is proposed in this paper.

2. Network Control

A computer network handling requests of users, so called network requests, can be interpreted as a set of interacting system - and user processes. Processes are the organizational entities within a computer associated with the execution of a

request and to which resources are allocated dynamically. We distinguish between user processes reflecting algorithms implemented by a user and system processes describing services provided by the computer network. The deadlock-free allocation of reusable resources is executed by special system processes. The internal allocation of resources within a computer is provided by local system processes.
The network control comprises the global coordination of the execution of network requests and is accomplished by network system processes.

Efficient network control is based on information about specification of requests, state of request execution and the state of the resources in order to decide on the mutual association of resources to processes applying a given optimizing strategy.

Processes may run in an independent manner or be coordinated if a request demands simultaneous use of several Hosts.
An organizational entity for the network control will be called an action. An action specifies a finite set of processes to be coordinated and the resources needed for their execution. It comprises the events, initialization and termination. The initialization of an action means the reservation of the required resources and the activation of the corresponding processes. Termination denotes the completion of processes and the releasing of resources no longer needed.

In the following a reliable network is assumed, i.e. all components of the network are supposed to work correctly.

3. A basic decentralized coordination mechanism

Coordinated execution of actions depends on the complexity of the requests and on the state of the resources. Before solving the general case of distributed resource management, a simplified problem shall be treated in order to develop a basic coordination mechanism.

We assume that each user request

- consists of exactly one action
- requires the allocation of all Hosts for exclusive use
- is executed without preemption and within a finite time.

The coordination of request execution can be achieved by providing each Host computer with a special component, a scheduler, which handles the resource allocation in cooperation with the other schedulers. For the cooperation, the schedulers have to exchange messages according to network wide accepted protocols depending on the structure of the network control: centralized or decentralized.

The principle of decentralized control is that each scheduler has to participate in deciding which of the waiting user requests has to be executed next. Complete agreement of all schedulers with respect to a decision must be achieved. Several strategies can be pursued. The strategy proposed consists of the strong coupling of scheduler activities; this ensures high reliability in service.

To define the basic coordination mechanism we assume that n Hosts with associated schedulers exist. Formally a scheduler can be described (cf. /4/, /7/) as a sequential automaton by the 7-tupel $(S, I_{in}, I_{ex}, O_{in}, O_{ex}, M, N)$ where $S = \{1,2,3,4\}$ denotes the set of states, $I_{in} = \{A_1, \ldots A_n, E\}$ resp. $I_{ex} = \{a,e\}$ represents the set of the internal resp. external input messages, O_{in} resp. $O_{ex} = \{d\}$ the set of internal resp. external output messages, $M : S \times (I_{in} \cup I_{ex}) \rightarrow S$ is the state transition function and $N : S \times (I_{in} \cup I_{ex}) \rightarrow O_{in} \times O_{ex}$ the output function.

Internal messages are used for the communication between schedulers. With the messages A_i, $i = 1(1)n$, the schedulers indicate mutually the agreement to execute that user request accepted by scheduler i. Also with message E it is shown that the schedulers have terminated the request execution. O_{in} contains the same message types as I_{in}.

External messages refer to the local interactions of a scheduler with users or with the local Host operating system. The message a ϵ I_{ex} indicates the existence of a waiting user request in the scheduler's Host. With message d ϵ O_{ex} a scheduler orders the start of the execution of a user request to the local Host which returns message e after termination of the request execution.

The graph in fig. 1 illustrates the activities of a scheduler. The elements of the state set S are represented by vertices, the functions M and N are defined by the directed edges marked according to the pattern:
external input, internal input/external output, internal output
("-" represents the empty message)

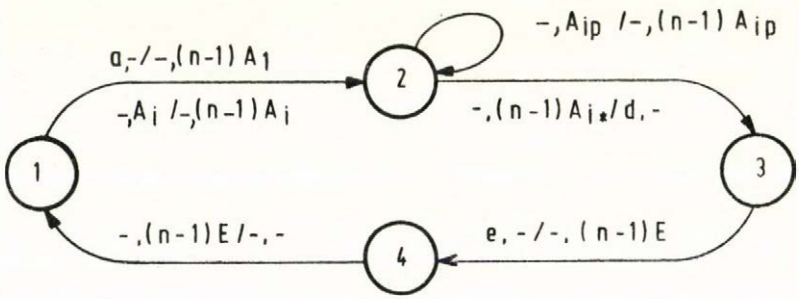


fig. 1: State transition diagram of scheduler 1 for the basic coordination mechanism

In detail a scheduler works as follows:

- A coordination cycle can only begin in state 1. It can be initiated by the scheduler (self-initiation) or by some other scheduler (external-initiation). After initiation a scheduler waits in state 2 for the agreements to its decision by the n-1 remaining schedulers.
- If the agreements of all the other schedulers to a user request (initiated by scheduler i*) have been received, a scheduler in state 2 performs the transition to state 3 as soon as it has declared its agreement to this request for its part.
- Since each scheduler is authorized to initiate coordination cycles, simultaneous initiations referring to different user

requests imply conflicts. To solve such conflicts user requests must be made distinguishable by means of unique priorities. A simple scheme is to use the identification scheme for the Hosts and the sequence of request generation within one Host.

- A supersession of user requests, being currently presented for agreement, by a user request A_{ip} of higher priority is only permitted when a scheduler is in state 1 or 2. A scheduler can only grant an externally initiated supersession if no n-1 agreement messages for a user request have arrived. It is allowed to propose a supersession only if it didn't send any agreements referring to a different user request and if it wasn't itself an initiator of a coordination cycle.
- In state 3 a scheduler orders the start of the user request execution and awaits the message e. After execution of the user request the scheduler sends the corresponding message E to the other schedulers and passes to state 4. The return to state 1 is performed if each of the other schedulers indicated, by message E, its readiness for a new coordination cycle.

Assuming a conflict-solving priority regulation for the user requests, this protocol provides for the transition of all schedulers into state 3 and prevents, as shown in /6/, that the schedulers being in state 3 have started the execution of different user requests. Since the coordinated return to state 1 is ensured, the protocol guarantees, in the case of a reliable computer network, the deadlock-free coordination of user request execution.

4. Distributed resource management by decentralized control

A general distributed resource management requires the schedulers to be equipped with scheduling algorithms (cf. /2/), which perform resource allocation and deallocation on the basis of allocation state information. For reliable scheduling this information has to be maintained identically by each scheduler.

As the algorithm must be the same for each scheduler, a deadlock-free resource allocation is achieved if the actual allocation information is kept consistent, i.e. each modification of that information has to be coordinated.

For simplicity we assume that an action must predeclare its needed subset of network resources (not necessary all Hosts as in 3). A preemption of an action in execution is not allowed.

To solve this general case of resource allocation the basic protocol can be used, i.e. if both events of action execution - initialization and termination - are handled in different coordination cyles /4/. The corresponding protocol for the coordination of the schedulers can be described by the set of states S = {1,2,3,3',4}, the internal messages $I_{in} = O_{in} = \{A_1, \ldots A_n, E\}$, the external messages $I_{ex} = \{a,e,v\}$, $O_{ex} = \{d\}$ and the functions M, N depicted by fig. 2.

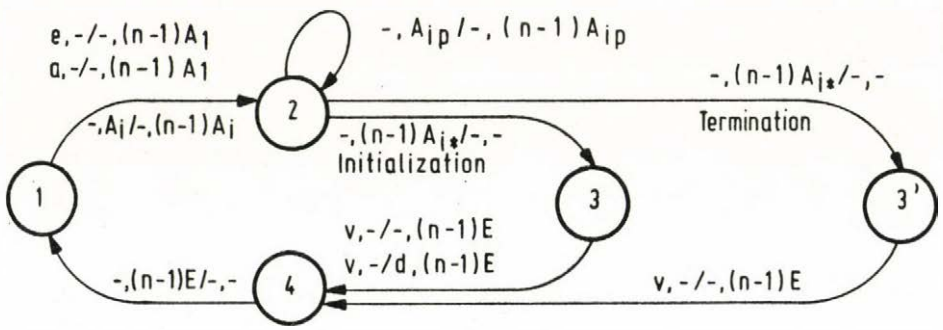


fig. 2: State transition diagram of scheduler 1 for the coordination of action execution

The meaning of the messages a,e,d is the same as in 3. The internal message v denotes the end of the scheduler's activities in state 3 resp. 3'. Messages A_i, i = 1(1)n, exchanged by the schedulers, indicate the agreement to execute the request accepted by scheduler i. A request in this case represents the demand either for the initialization or for the

termination of an action, which has to be specified by the message too.

A coordination cycle in this general case doesn't concern the coordination of request handling as in 3., but the coordination of initialization or termination of actions. This leads to the extension of the set of states:
In state 2 the initialization of an action implies a transition to state 3, a termination, to state 3'.

To solve conflicts during the agreement phase and to augment utilization of resources, termination of an action has higher priority than an initialization.

In state 1 and 2 a scheduler works the same as described in 3. In state 3 (consequence of the initialization of an action) a scheduler allocates resources to the action, updates the state of resources and starts the action (if and only if the computer corresponding to the scheduler has to be involved in the execution of that action). On the other hand, in state 3' the state of the resources has to be updated only.

After having exchanged messages E indicating the end of the activities in state 3 or in state 3', all schedulers perform a transition to state 1 and are ready to start a new coordination cycle.

To achieve a deadlock-free coordination of requests (under the assumption of a reliable network), a request for the initialization of an action can only be agreed to if the resources demanded by that action are actually available for allocation. This must be tested in state 1 by inspecting the allocation state information.

We can consider the execution of requests consisting of sets of (dependent or independent) actions using the protocol described above.
A requirement for the protocol to be applicable is that a detailed specification of the resources needed for action

execution exists.

5. Implementational aspects

5.1. Integration of schedulers in the Host computers

An obvious way of implementing the schedulers in a computer network is to completely integrate the corresponding activities in the Host computers. This implementation assumes suitable properties of the communication system for the exchange of messages between the processes (e.g. users, schedulers) in the network. The concept of connection oriented communication is regarded as the most appropriate of the existing alternatives /1/. According to this concept a directed logical connection, which supports the transmission of an arbitrary number of messages between processes, is established before transmitting messages. Communication primitives are needed to create/ delete a logical connection and to send/receive messages.

Under this assumption the scheduler may be structured as shown schematically in fig. 3 (cf. /6/).

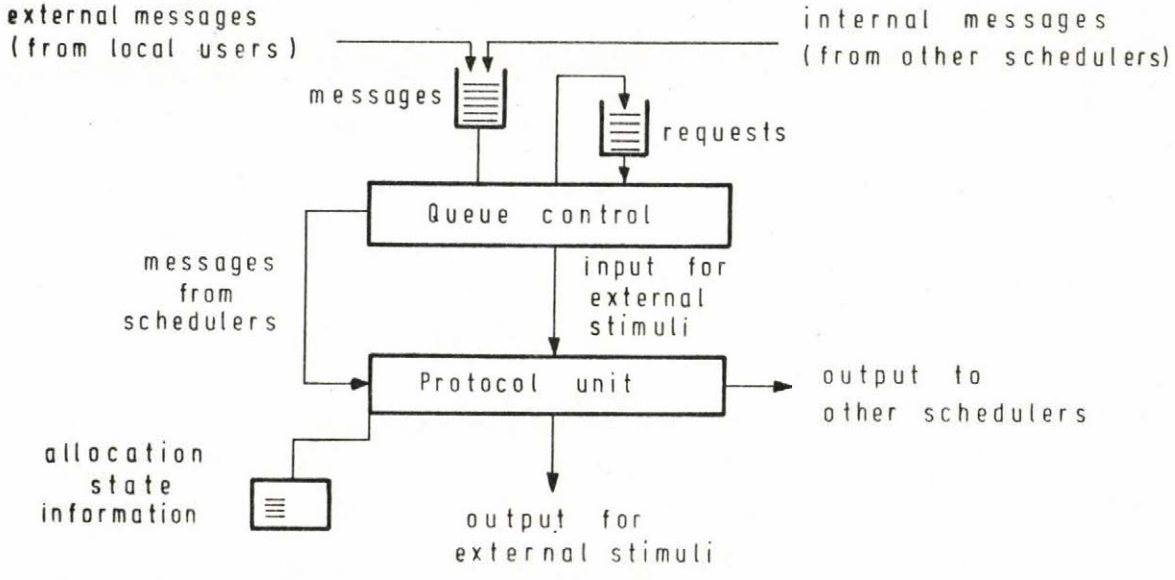


fig. 3: Components of a scheduler

The queue control mechanism manages the message queue and the request queue: As a consequence of an "initialization message" a request (on performing a coordination cycle) is created and placed in the request queue; after having terminated action execution, the corresponding request is eliminated; information (e.g. messages) is passed to the protocol unit being interpreted there.

The protocol unit has to determine the next activity of the scheduler, considering the actual state of the protocol. In particular, messages for the other schedulers or to the local user resp. operating system (external stimuli) are created. Decisions of the protocol unit are enabled using a table of the actual state of resource allocation (allocation state information).

5.2. Realization of schedulers applying special hardware

The complete integration of the schedulers in the Hosts implies a considerable overhead due to activities for the transfer of messages between schedulers. If the network is not reliable, the schedulers in addition to the management of the coordination cycles also have to survey most of the protocol activities by means of time-outs.

In many computer networks, message exchange between Hosts is supported by special computers, communication processors. We suppose that each Host is linked to exactly one of these communication processors and vice versa.

An approach to reduce the load of Hosts is to implement an administrator A_i in $Host_i$ and a secretary S_i in the corresponding communication processor CP_i. The secretaries possess a structure analoguous to the schedulers of 5.1.

In this case the steps of request execution are as follows:

- a request R, originated from $Host_j$, is passed to the local administrator A_j,
- A_j assigns a (unique) priority to R and transmits the request to its secretary S_j,
- S_j discusses a possible execution of R, communicating with the secretaries on the other communication processors,
- if request execution is accepted by the secretaries, the corresponding administrators are advised to cause this execution,
- the administrators indicate the termination of request execution to the secretaries, which finish the request execution cycle.

As the control of time-outs can be executed by the secretaries, the values of time-outs can be chosen more appropriate than in the case of the implementation described in 5.1.

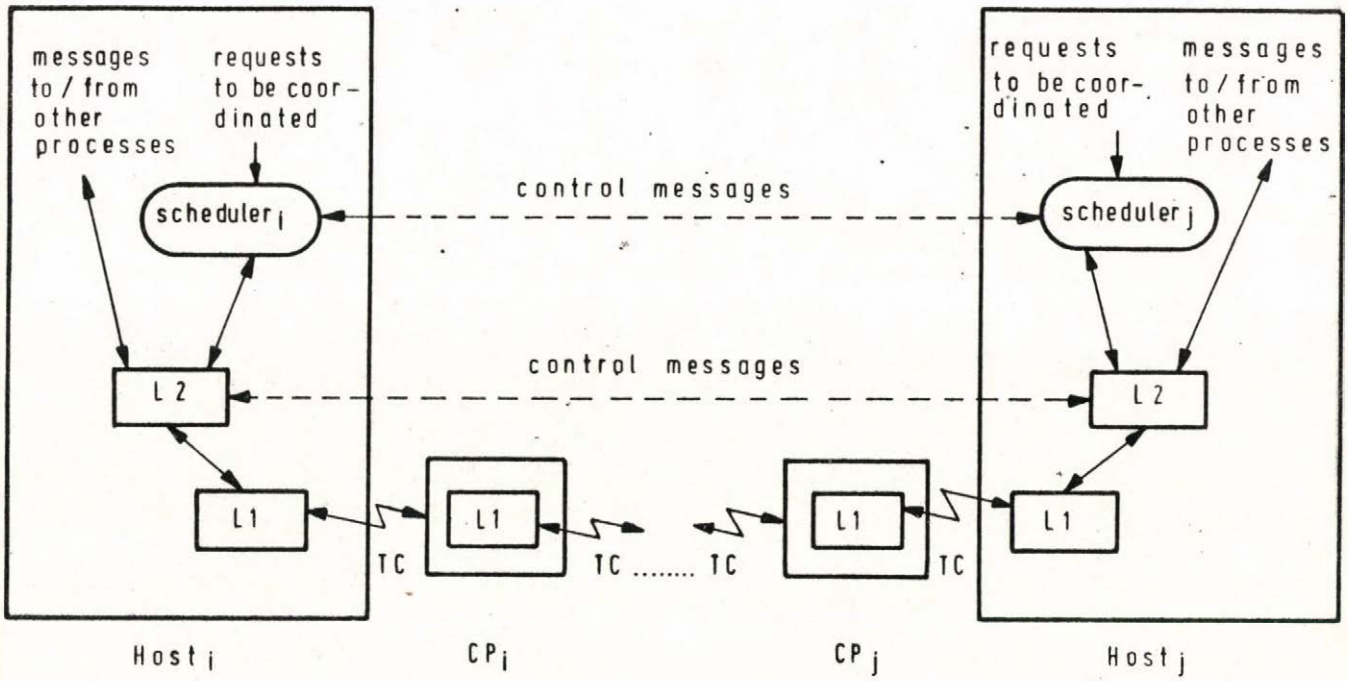
6. Performance analysis by means of simulation

6.1. Models for computer networks

To obtain a feeling for the qualitative and quantitative difference between the implementations proposed in 5.1. and 5.2., a simulation-based modeling system*) of an existing computer network /8/ was used; it is implemented in SIMULA /3/. The simulation model was based on a suitable model for communication systems in computer networks. These communication systems usually are structured in different layers, corresponding to the hierarchy of different levels of communication protocol (see fig. 4).

---

*) The modeling system was developed at the Institut für Datenverarbeitung in der Technik (Karlsruhe) at the request of the Hahn-Meitner-Institut (HMI - Berlin); it supports a detailed investigation of the message flows in the layer-structured protocol hierarchy of the HMI-computer network /8/.



 real information flow

 virtual information flow

fig. 4: Integration of schedulers in the protocol hierarchy of a computer network
(CP: Communication processor; TC: Transmission channel)

The meaning of the protocol layers may be:

- L1: line procedure, regulating data transmission between two physically connected computers
- L2: packet level, regulating data exchange between Host computers.

A protocol layer of the communication system is represented by a protocol module in a computer (Host, communication processor). Modules of identical layers exchange control messages (e.g. in order to establish flow control) whereas directly coupled protocol modules of different layers within

a Host exchange commands or messages. As in existing computer networks, the protocol modules of the simulation model are separated into sending and receiving parts to consider different priorities or buffer sizes for the sending or receiving activities.

In the hierarchy of communication protocols, schedulers can be viewed as "users" of the communication system. From the point of view of the communication system, schedulers are equivalent to user processes exchanging data.

To model the competition of the protocol modules and the scheduler or the secretary for the resources within a computer (Hosts, communication processors) the concepts detailed in /5/ were applied.

6.2. Experimental results

The comparison of the scheduler implementations detailed in 5.1. (version I) and 5.2.(version II) are based on a configuration of four Host computers each of which is connected to a communication processor via a channel-channel link ($170 \frac{\text{K Bytes}}{\text{sec}}$). The communication processors themselves are completely interconnected by high speed serial connections ($17 \frac{\text{K Bytes}}{\text{sec}}$). Service times for software activities were chosen according to an existing computer network (/8/).

Two different types of information were transferred by the communication system

- control data between the schedulers resp. secretaries (length: 10 Bytes)
- messages between other processes (length: arbitrarily chosen in the interval [100, 500] Bytes) (background stream).

The particular objective of the experiments was the investigation of the influence of the background stream and request intensity on Host and communication processor utilizations. Measurements

were carried out in the stationary state. The experiments were considering different arrival rates "a" of messages (between user processes) and various arrival rates "b" of requests (for coordination).

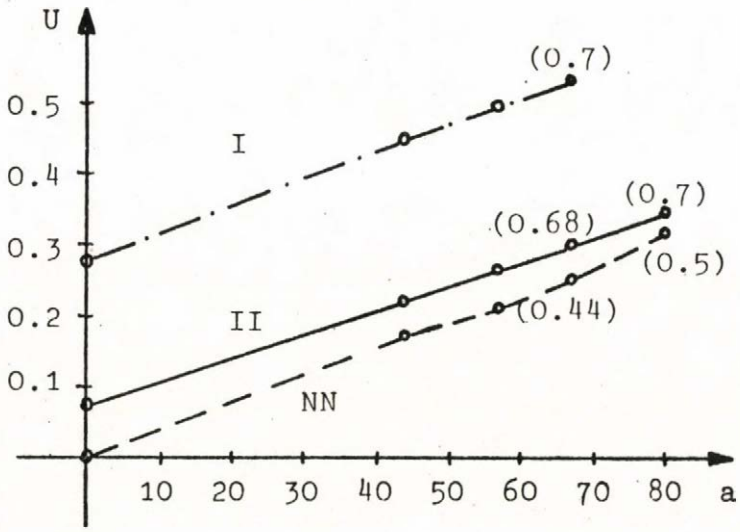


fig. 5: Host utilization U in dependance of (message) arrival rate a [1/sec] for versions I and II (b = 4 [1/sec] constantly). NN describes the system without any requests for executing coordination cycles, i.e. b = 0. Utilization of communication processors is marked in parentheses.
(note: version I is not stationary for a = 80).

Fig. 5 shows the considerable superiority of version II in comparison to version I. In version II, utilization of Host computers as well as those of communication processors is significantly smaller. Furthermore the stationary state of the system is maintained for larger arrival rates of (user) messages.

7. Conclusion

In this paper coordination mechanisms for the decentralized control of request execution have been proposed. Two methods for a possible realization of those protocols in a computer network have been demonstrated. Performance evaluations for possible implementations were obtained, applying a simulation model of an existing computer network.

The coordination protocols introduced may be used e.g. to support integrity in distributed data bases or to coordinate activities in distributed systems for control of technical processes. As shown in /4/, these coordination protocols can be extended to fault-tolerant protocols, meeting the requirements of real computer networks with inherently unreliable components.

References:

/1/ Akkoyunlu, E., Bernstein, A., Schantz, R.
Interprocess communication facilities for network operating systems
Computer, June 1974, p. 46

/2/ Coffmann, E.G., Denning, P.J.
Operating Systems Theory
Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973

/3/ Dahl, O.J., Myhrhaug, B., Nygaard, K.
SIMULA 67 - Common Base Language
Norwegian Computing Center, Forskningsveien 1B, Oslo, 1968

/4/ Drobnik, O.
Strukturmodelle dezentralisierter Kontrolle in Mehrrechnersystemen
Rechnernetze und Datenfernverarbeitung, Fachtagung der GI und NTG, Aachen, 1976

/5/ Drobnik, O., Wolfinger, B., Holler, E.
Simulation of task interaction and resource allocation in communicating computers
SIMULA Users' Conference, Noordwijkerhout, 1976

/6/ Holler, E.
Koordination kritischer Zugriffe auf verteilte Datenbanken in Rechnernetzen bei dezentraler Überwachung
Gesellschaft für Kernforschung Karlsruhe, KFK-1967, April 1974

/7/ Le Moli, G.
A Theory of Colloquies
1. European Workshop on Computer Networks, Arles, 1973

/8/ Strack-Zimmermann, H.W., Schrödter, H.D.
The Hahn-Meitner-Institut Computer Network
Rechnernetze und Datenfernverarbeitung, Fachtagung der GI und NTG, Aachen, 1976

Decentralizált hálózatok tervezése és szimulációja

Wolfinger, B., Drobnik, O., Holler, E.

A dolgozat decentralizált számitógép hálózatok koordinálásának megbizhatósági problémájával foglalkozik. Ismertet egy ütemezési módszert, és leirja annak hatékonysági vizsgálatát.

Проектирование и имитация децентрализованных сетей

Б. Волфингер, О. Дробник, Е. Холлер.

В работе исследуются проблемы надежности управления децентрализованными сетьями вычислительных машин. Описывается некоторый метод распределения ресурсов и анализируется эффективность метода.

ОБ ОРГАНИЗАЦИИ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА В ОДНОМ КЛАССЕ ПАКЕТОВ ПРИКЛАДНЫХ ПРОГРАММ НА ЕС ЭВМ

И.В Сергиенко, И.Н. Парасюк

Создание эффективных пакетов прикладных программ /ППП/ является одним из путей повышения эффективности применения вычислительной техники. Скорость и точность решения прикладных задач с помощью ППП, возможность решения с их помощью сложные задачи являются одним из показателей качества и эффективности ППП. Эти показатели существенно зависят от способов организации проблемного математического обеспечения /ПМО/ ППП и процесса выполнения программных модулей /ПМ/ ПМО ППП.

Одним из этапов организации ПМО ППП является модульный анализ класса прикладных задач, т.е. создание так называемой системы программных модулей из множества алгоритмов решения задач данного класса /аналог модели предметной области [3] /. Эти вопросы представляют самостоятельный интерес и уже, в частности, изучались в работах [2, 3, 4] .

В настоящей статье систематизируются различные способы структурирования ПМ ПМО ППП и их диспетчирование в процессе решения задач данного класса в зависимости от некоторых требований к ресурсам ЭВМ. При этом, говоря о ППП, условимся понимать, что речь идет о ППП, строящихся на ЕС ЭВМ с использованием существующих версий операционных систем (PCP, MFT, MVT) [5] , а их ПМО представлено модульным графом G^x [4] . Если специально не оговорено, какая из версий операционной системы имеется в виду, будем понимать, что сказанное относится к любой из выше перечисленных версий.

Итак, пусть задан модульный граф G^x , представляющий структуру класса задач и методов их решения [4].

$$G^x = G^x (Z, M, F),$$

где Z - конечное множество вершин, соответствующее множеству задач $Z = \{z_1, z_2, \ldots, z_{|Z|}\}$, представляющих заданный класс задач Z;

M - конечное множество вершин, соответствующих ПМ

$$M = DUOUE,$$

где D, O, E - соответственно конечные множества модулей типов данных, операционных и эргативных модулей;

F - информационные дуги, отображающие информационную взаимосвязь между ПМ;

G^{*} - ярусно-параллельный информационный граф.

Рассмотрим структурирование ПМ и их диспетчирование в зависимости от различных характеристик класса задач - ограничений на память и на время при решении задач этого класса. При этом для определенности будем предполагать, что ППП работает в режиме интерпретации [7].

Обозначим V - память, выделенная ППП для выполнения модулей $M_i \in M$; V_i - память, необходимая для выполнения модуля M_i;

$T(Z_j)$, T_i - соответственно время /коммерческое/ решения задачи Z_j и выполнения модуля M_i;

$\tau(Z_j)$, τ_i - соответственно время простоя центрального процесса при решении задачи Z_j и при выполнении модуля M_i;

$t(Z_j)$, t_i - соответственно чистое время решения задачи Z_j и чистое время счета модуля M_i;

$\tilde{T}(Z_j)$, $\tilde{T}_i$ - соответственно допустимое время решения задачи Z_j и допустимое время выполнения модуля M_i.

I. Пусть $\max V_i \leq V$, $T_i \leq \tilde{T}_i$ $(i = 1,2,\ldots,|M|)$. Следовательно $T(Z_j) \leq \tilde{T}(Z_j)$ $(j = 1,\overline{|Z|})$. Иными словами память и время фактически нелимитируют. Такие ограничения обычно присущи несложным задачам, решающимся на общих основаниях в общем потоке задач.

I.I. Структурирование модулей.

В этом случае целесообразно создавать модули простой структуры [5].

Модули простой структуры являются наиболее эффективными в смысле скорости выполнения. Это связано с тем, что они содержат все необходимые для своего выполнения операторы, т.е. динамически не используют других модулей и поэтому не используют таких макрокоманд загрузки как **LINK, LOAD, XCTL, ATTACH.** Модули этой структуры могут содержать подпрограммы, обращение к которым осуществляется с помощью макрокоманы **CALL** /или с помощью операторов перехода/ и таким образом модули этой структуры выполняются без участия супервизора.

Модули простой структуры как и модули любой другой структуры начинаются макрокомандой **SAVE** и завершаются макрокомандой **RETURN.**

1.2. Диспетчирование модулей.

Пусть требуется решить задачу $z_j \in Z$, исходные данные которой принадлежат к типу $D_k \in D$. Тогда совокупность модулей, упорядоченное выполнение которых приводит к решению данной задачи определяется некоторым подграфом G_{jk}, который используя обозначения [4] можно записать соотношением

$$G_{jk} \equiv H[A_k(z_j)] = \bigcup_{v=1}^{m_{jk}} \pi_k^v(e_j). \qquad (1)$$

Если требуется решить подмножество задач $Z_k \subseteq Z$, исходные данные которых принадлежат к типу D_k, то подграф $G^*_{Z_k} \subseteq G^*$, пред-

ставляющий состав вычислительной схемы обобщенного алгоритма решения задач в терминах и обозначениях работы [4] определяется соотношением

$$G^{*}_{Z_k} \equiv H[A\ (Z_k)] = \bigcup_{j=1}^{|Z_k|} H[A_k(z_j)] \qquad (2)$$

Анализ, информационной взаимосвязи программных модулей входящих в подграфы G_{jk}, $G^{*}_{Z_k}$ позволяет определить порядок выполнения модулей, который можно описать правилами:

1^{o}. Модули одного яруса выполняются в произвольном порядке;

2^{o}. Модули m-го яруса можно выполнить после выполнения смежных с ними модулей (m-1), (m-2),..., 0 ярусов.

2. Пусть $\max_i V_i > V$, $T_i \quad \tilde{T}_i$ $(i = 1,2,\ldots,|M|)$.

В этой ситуации $T(Z_j) \leqslant \tilde{T}(Z_j)$ $(j = 1,2,\ldots,|Z|)$, однако в множестве модулей M существуют модули, которые не могут быть выполнены в памяти объемом V. Пусть это будут модули M_{i_1}, M_{i_2},.., M_{i_η}.

Эти ограничения обычно характеризуют задачи, которые отличаются от описанных в п. 1. большими объемами перерабатываемых данных и более громоздкими алгоритмами.

2.1. Структурирование модулей.

Рассмотрим вкратце основные свойства оверленовой структуры /структуры с перекрестиями/.

Модуль оверленовой структуры делится на сегменты, которые могут совместно использовать одну и ту же область основной памяти, т.е. могут выполняться в одной и той же области. Исключение составляет главный т.е. корневой сегмент модуля, который всегда находится в памяти во время выполнения других сегментов

этого модуля. Смену сегментов осуществляет супервизор, в связи с чем при выполнении модуля оверлейной структуры происходит потеря времени на организацию смены сегментов. Сегменты, которые совместно используют одну и ту же область памяти /но не принадлежат одному и тому же пути/ называется исключающими /но не могут находиться одновременно в памяти, и, как правило, имеют одинаковые относительные имена/. Сегменты, которые могут находиться в памяти одновременно называются включающими. Обмен памяти между исключающими сегментами осуществляется через область памяти старшего включающего сегмента.

Размещение /планирование/ программных секций в оверлейном модуле можно сделать при помощи предложений редактора связей операторной системы [5] **INCLUDE, INSERT,** а также путем размещения объектных колод сегментов /секций сегментов/ относительно предложений **OVERLAY.** Программные модули оверлейной структуры бывают однообластные и многообластные. Модуль однообластной оверлейной структуры может быть представлен в виде оверлейного дерева, корнем которого является корневой сегмент. Если некоторый сегмент программного модуля требуется различным секциям из различных петей и нет возможности /из-за ограничений на память/ разместить этот сегмент в корневой сегмент, необходимо поместить этот сегмент в отдельную область памяти. Это можно сделать при помощи управляющего предложения редактора связей **OVERLAY** и слова **REGION.**

При планировании памяти для модулей оверлейной структуры необходимо учитывать память оверленого супервизора, таких его таблиц как таблица сегментов **(SEGTAB),** таблицы входов **(ENTAB),** список примечаний **(NOTELST),** которые помещаются в оверлейный модуль.

Таким образом в рассматриваемом случае каждый модуль $M_{i_v} \in \{M_{i_1}, \ldots, M_{i}\}$ целесообразно представить в виде модуля оверлейной структуры, т.е. каждый модуль M_{i_v} должен быть представлен в виде функционально связанных фрагментов сегментов

M_i^j ($j = 1,2,\ldots,m_{i\nu}$) таких, что $V_{i\nu}^j \leq V - V_{i\nu}^1$. Причем, разбиение модулей $M_{i\nu}$ на сегменты $M_{i\nu}^j$ будем считать оптимальным, если $m_{i\nu} = \min_{\ell} (m_{i\nu})$, где $m_{i\nu}$ - число сегментов в модуле $M_{i\nu}$, созданных ℓ -м способом; $V_{i\nu}^1$ - память, занимаемая корневым сегментом.

Замечание. Иногда, когда, например, модули $M_{i\nu}$ имеют сложную структуру, бывает затруднительно представить из в виде модулей оверлейной структуры. В таких случаях модули $M_{i\nu}$ можно представить в виде модулей динамически-последовательной структуры.

Модуль динамически-последовательной структуры в процессе своего выполнения динамически вызывает модули, которые необходимо выполнить для нормального завершения его работы. Вызов, загрузку и наполнение этих модулей осуществляетсупервизор с помощью одной из макрокоманд: **LINK, XCTL** в качестве одного из параметров **(EP)** которых является имя вызываемого модуля. Если некоторые модули используются многократно в процессе решения задачи, то для их загрузки и удаления из памяти целесообразно использовать макрокоманды соответственно **LOAD** и **DELETE**. После загрузки модуля с помощью макрокоманды **LOAD** передать ему управление можно с помощью макрокоманды **LINK** или операторов передачи управления.

Модули, использующиеся в качестве составных частей модулей динамическо-последовательной структуры, могут быть модулями простой или оверлейной структуры.

2.2. Диспетчирование модулей.

Определение состава вычислительных схем $H[A_k(Z_j)]$, $H[A^*(Z_k)]$ осуществляется соответственно по формулам (1), (2), а порядок выполнения модулей определяется по пра вилам 1^o, 2^o.

3. Пусть $\max_i V_i \leq V$, $T_i > \tilde{T}_i$ ($i=1,2,\ldots,|M|$) и следовательно $T(Z_j) > \tilde{T}(Z_j)$ ($j=1,2,\ldots,|Z|$). Такие ограничения, обычно, при-

сущи классу несложных задач, решающихся в реальном масштабе времени, либо теряющих слишком много времени на ожидание ресурсов так, что процесс их решения нельзя признать удовлетворительным.

К этому случаю относятся и классы задач, все задачи и следовательно все модули M_i, которые не обязательно характеризуются временами $T(Z_j) > \tilde{T}(Z_j)$, $T_i > \tilde{T}_i$. Будем для определенности полагать, что существуют задачи $\{Z_1, Z_2, \ldots, Z_p\} \in Z$, а следовательно и некоторых модулей $\{M_{i_1}, M_{i_2}, \ldots, M_{i_\eta}\} \leftarrow M$ такие, что $T(z_{j_\mu}) > \tilde{T}(z_{j_\mu})(\mu = 1,2, \ldots, p)$, $T_{i_\nu} > \tilde{T}_{i_\nu}(\nu = 1,2, \ldots, \eta)$.

3.1. Структурирование модулей.

При структурировании модулей $M_{i\nu}(\nu = 1,2,\ldots, \eta)$ нужно иметь в виду, что $T(z_{j_\mu}) = t(z_{j_\mu}) + \tau(z_{j_\mu})$, $T_{i_\nu} = t_{i_\nu} + \tau_{i_\nu}$ и, следовательно, величины $\tau(z_{j_\mu}) > 0$, $\tau_{i_\nu} > 0$ $(\mu = 1,2, \ldots, p, \nu = 1,2, \ldots, \eta)$. При этом возможны два случая:

3.1.1. $(\forall_{z_j} \in Z)[t(z_j) < \tilde{T}(z_j)]$.

3.1.2. $(\forall_{z_j} \in Z)\exists\, z\xi_1, \ldots, z_{\xi\varphi}[t(z_{\xi 1}) \geqslant T(z_{\xi 1}) \geqslant T(z_{\xi 1}) + t(z\xi\varphi) > \tilde{T}(z_{\xi\varphi})]$.

Случай 3.1.2. не представляет интерес в рамках проблематики настоящей статьи, так как он связан с поиском по существу новых методов решения задач $z_{\xi 1}, z_{\xi 2}, \ldots, z_{\xi\varphi}$

Случай 3.1.1. представляет практический интерес так как можно предпринять попытку минимизации составляющих $\tau(z_{\xi 1}), \ldots, \tau(z_{\xi\varphi})$

Задача минимизации составляющих $\tau(z_{\xi 1}), \ldots, \tau(z_{\xi\varphi})$ сводится к минимизации составляющих τ_i программных модулей. Это возможно в рамках операционной системы **MVT** с помощью макрокоманд **ATTACH, DETACH, WAIT, POST** и блоков управления событиями **ECB**. С помощью этих средств модули M_{k_i} $(i = 1,2,\ldots, \eta)$ необходимо

представить как модули динамически-параллельной структуры[5]. Модуль этой структуры во многом похож на модуль динамически-последовательной структуры. Отличие состоит в том, что модуль динамически-параллельной структуры позволяет организовать выполнение модулей, которые ожидают завершения некоторых событий, параллельно. Динамически-параллельные структуры возможно организовать лишь в операционной системе **MVT** с помощью макрокоманды **ATTACH**. Модуль, который выдает макрокоманду **ATTACH** называется порождающим /иначе порождающая задача/, а модули, которые образованы этой макрокомандой как отдельные задачи, называются подзадачами. Уничтожение порожденных подзадач осуществляется макрокомандой **DETACH**. Задачам и подзадачам присваивается диспетчерский приоритет /число из диапазона 0-255/, который существенно влияет на порядок выполнения модулей: в каждый момент времени управление получает модуль, который имеет наивысший диспетчерский приоритет и находится в состоянии готовности к выполнению /более подробно об этом см. [5]/.

3.2. Диспетчирование модулей.

При решении вопросов об автоматизации процессов планирования порядка выполнения ПМ, структурирование как динамически-параллельные модули, возможны два случая.

3.2.1. $(\forall_{M_i} \in M)(T_i \leqslant \tilde{T}_i)$.

3.2.2. $(\forall_{M_i} \in M)\exists\,(M_{i_1}, \ldots, M_{i_\eta})[(T_{i_1} > \tilde{T}_{i_1}) + (T_{i_\eta} > \tilde{T}_{i_\eta})]$.

В условиях случая 3.2.1. планировать порядок выполнения модулей целесообразно по аналогии с п. 1.2.

Условия случая 3.2.2. означают, что путем структурирования модулей M_{i_μ} как модулей динамически-параллельной структуры не удалось уложить время некоторых программных модулей в заданные пределы. Возникае задача сведения к минимуму суммарного време-

ни выполнения программных модулей за счет составляющих $\tilde{c}$. Отсюда следует, что выполнение модулей M_{i_μ} необходимо организовать динамически-параллельно. Как уже отмечалось, это возможно реализовать в рамках операционной системы **MVT** с помощью средств: **ATTACH, DETACH, WAIT, POST, ECB** путем соответствующих доработок элементов ведущей программы, реализации заданий ППП.

Для изучения этой задачи рассмотрим более общий случай, когда $M_\eta = \{M_{i_1}, \ldots, M_{i_\eta}\} = M$, т.е. когда все модули множества M обладают свойствами $T_i > \tilde{T}_i$, $V_i < V$ /после процесса структурирования с учетом п. 3.1./.

Требуется свести к минимуму суммарное время выполнения модулей вычислительной схемы за счет использования времени ожидания ресурсов в процессе решения задач.

Совокупность программных модулей, выполнение которых приводит к решению отдельных задач $Z_i \in Z$ или некоторого подмножества задач $Z_k \in Z$, их взаимосвязь как и в предыдущих случаях можно записать соотношениями /1, 2/. Остается открытым вопрос о планировании порядка выполнения модулей в каждом из ярусов гафа G^* с целью минимизации суммарного времени выполнения.

Пусть M_1^j , $M_2^j, \ldots,$ $M_{\eta i}^j$ - совокупность вершин-модулей подграфа $G^*_{Z_k}$ /обозначим ее $|M_i^j|$ / j-го яруса.

Требуется построить такую последовательность $\{M_i^j\}$, чтобы суммарное время выполнения модулей $M_1^j, \ldots,$ $M_{\eta j}^j$ за счет использования составляющих $\tilde{c}_i^j$ было минимально.

В предположениях, что имеют место неравенства:

$$(V_i = 1,2, \ldots, \eta_j)(V_l = 1,2, \ldots, \eta_j)[(i \neq l) \rightarrow (T_i^j \geqslant \tau_l^j)] \qquad (3)$$

суммарное время выполнения модулей j-го яруса как модулей динамически параллельной структуры можно представить соотношением:

$$T^j = \sum_{i=1}^{\eta_j} (T_i^j - \tau_i^j) + 2\tau_{\eta_j}^j = \sum_{i=1}^{\eta} t_i^j + 2\tau_{\eta_j}^j \quad (4)$$

Нетрудно видеть, что величина T^j достигает своего минимума, когда величина $\tau_{\eta_j}^j$ /т.е. время ожидания последнего в последовательности модуля/ минимальна. Отсюда следует, что при выполнении условий (3) в качестве искомой последовательности $\{M_i^j\}$ можно взять любую последовательность модулей j-го яруса, последний элемент которой характеризуется минимальной величиной τ_i^j .

На практике довольно редко удается получить величины τ_i^j в чистом виде. В качестве их оценок могут служить величины $C_i^j . T_i^j$ где C_i^j - коэффициент ожидания, характеризующий долю времени затраченного модулем M_i j-го яруса на ожидание ресурсов или (u) выполнение операций ввода-вывода. При выборе значений коэффициентов ожидания C_i^j целесообразно учитывать потери времени на организацию совмещения.

условия (3) не являются слишком жесткими ограничениями. Легко показать, что к этому случаю можно еще свести и другие совокупности $[M_i^j]$. Например, если из совокупности $[M_i^j]$ удается построить последовательность такую, что $\tau_{\eta j}^j = \min_{1 \leqslant i \leqslant \eta_j} \tau_i^j$, а для каждой пары (M_i^j, M_{i+1}^j)модулей этой последовательности выполняется неравенство $\tau_i \leqslant T_{i+1}^j$, то эта последовательность тоже будет оптимальная в смысле минимизации величины T^j.

В общем случае затруднительно получить аналитическое выражение для величины T^j и, следовательно, указать, как лучше строить последовательность $\{M_i^j\}$. Однако исследования /например, последовательно анализируя случаи для $\eta_j = 2, 3, 4, .../$ показывают, что для получения значения близкого к минимальному значению величины T^j можно построить последовательность $\{M_i^j\}$ в порядке убывания величин τ_i^j . Оптимальную последовательность $\{M_i^j\}$ можно получить с помощью метода полного перебора вариантов.

Таким образом в общем случае один из алгоритмов выполнения программных модулей подграфов $G^{*}_{Z_k}$ в ППП, работающих на мультипрограммной вычислительной системе, с цельюминимизации времени решения задач состоит в следующем:

1^{*}. Все модули, принадлежащие подграфу $G^{*}_{Z_k}$ поярусно упорядочиваются в искомые последовательности $\{M_i^j\}$ /j - номер яруса, i - номер модуля в ярусе/.

2^{*}. Модули последовательности $\{M_i^j\}$ выполняются поярусно.

3^{*}. Ожидающий модуль M_i^j подключается к выполнению модуля M_{i+1}^j последовательности $\{M_i^j\}$ при условии, что модуль M_{i-1}^j выполнен, либо не может быть выполнен из-за того, что продолжает находиться в состоянии ожидания.

4. Пусть $V_i > V$, $\tilde{T}_i > T_i$ $(i = 1,2,\ldots,|M|)$, $T(Z_j) > \tilde{T}(Z_j)$ $(j = 1,2,\ldots,|Z|)$.

Эта ситуация может возникать для так называемых больших задач, решаемых в реальном масштабе времени /или в условиях близких к этому/.

4.1. Структурирование модулей.

Каждый модуль $M_i \in M$ $(i = 1,2,\ldots,|M|)$ структурируется как модуль оверлейной структуры /по аналогии с п. 2.1./, кроме того, используются средства операционной системы **MVT (ATTACH, DETACH, WAIT, POST, ECB)** распараллеливается вычислительный процесс так, чтобы величина τ_i была минимальна для данного модуля M_i.

4.2. Диспетчирование модулей.

Предположим, что после структурирования модулей согласно п.4.1. в множестве M существуют модули $(M_{i_1},\ldots, M_{i_\eta})$, для которых $\tau_{i\mu}(\mu = 1,\ldots,\mu)$ и следовательно, для некоторых задач, нап-

ример, $(z_{j_1}, \ldots, z_{j_p})$

$$T(z_{j_1}) > \tilde{T}(z_{j_1}, \ldots, T(z_{j_p}) > \tilde{T}(z_{j_p}).$$

В этом случае планирование порядка выполнения программных модулей осуществляется согласно рекомендаций п. 3.2.2.

Следует отметить еще одну возможность экономии времени при решении прикладных задач с помощью ППП на ЕС ЭВМ с использованием операционной системы **MVT**. Это касается процесса структурирования ПМ ПМО ППП, сущность которого состоит в следующем. Среди всех ПМ ПМО ППП необходимо выделить также ПМ, которые многократно используются в прецессе решения задач. После того, как такие модули отобраны, с помощью редактора связей их целесообразно структурировать как повторно используемые [5], что позволяет сэкономить время, затраченное на повторную загрузку этих модулей в рабочую область.

Выше были рассмотрены вопросы организации вычислительного процесса в ППП, ориентированные на однопроцессорную ЭВМ семейства ЕС. Особый интерес приобретают затронутые вопросы в пакетах программ, ориентированные на комплекс ЕС ЭВМ. Известно, что ЭВМ ЕС 1030 и ЕС 1050 могут быть объеденены в вычислительный комплекс, обладающий уже двумя процессорами. Управление этим комплексом осуществляет специально сгенерированная операционная система являющаяся расширением ОС. Таким образом, кроме отмеченных выше результатов появляется возможность применить имеющиеся к настоящему времени результаты по распараллеливанию вычислитильного процесса на комплексе ЭВМ /см. например [8] / и к пакетам прикладных программ.

ЛИТЕРАТУРА

1. Серниенко И.В., Парасюк И.Н., Тукалевская Н.И.
Автоматизированные системы обработки данных.
Изд-во "Наукова думка", К., 1976.

2. Коновалова А.Н., Яненко Н.М.
Модульный принцип построения программ как основа создания прикладных программ решения задач механики сплошной среды. - В сб. "Комплексы программ математической физики". Новосибирск, СО АН СССР, 1972.

3. Там Б.Г., Тыугу Э.Х.
О создании проблемно-ориентированного программного обеспечения.
Ж. "Кибернетика", № 4, 1975.

4. Сергиенко И.В., Парасюк И.Н.
Некоторые вопросы разработки и исследования одного класса универсально-специализированных систем обработки данных.
Ж. "Кибернетика", № 6, К., 1973.

5. Операционная система IBM/360. Супервизор и управление данными, перевод с английского под ред. А.И. Илюшина.
Изд-во "Советское радио", М., 1973.

6. Сергиенко И.В., Парасюк И.Н., Тукалевская Н.И.
Универсально-специализированная система обработки данных /система УСОД/.
Ж. УСиМ, № 2, 1974.

7. Сергиенко И.В., Стукало А.С., Парасюк И.Н., Кудринский В.С.
Об одном классе пакетов статистической обработки данных: Труды Международной конфереиции "Структура и организация пакетов программ".
Изд-во ВЦ АН СССР, М., 1976.

8. Пашкеев С.Л.
Основы мультипрограммирования для специализированных вычислительных систем.
Изд-во "Советское радио", М., 1972.

Egy alkalmazási programcsomag-osztály ütemezési algoritmusáról ESZR számítógépekre

I. V. Szergijenkó, I. N. Paraszjuk

A szerzők korábbi kutatásai eredményeinek konkrét alkalmazása ESZR/OS operációs rendszer vezérlete alatt működő alkalmazási programcsomagok optimális ütemezési algoritmusának meghatározására.

On the scheduling for a class of program packages for Ryad machines

I. V. Sergiyenko, I. N. Parasyuk

In this paper the authors apply their previous results for solving a concrete problem - the problem of optimal scheduling for program pachages running under the control of OS operating system.

ON SOME FEATURES OF PAGING ALGORITHMS

by D. Bárdossy /VIDEOTON Research Institute/
and A. Iványi /Eötvös Loránd's University/

INTRODUCTION

The majority of the papers on paging - for example the works of Denning, King, Arató, Kogan, Chu and others - deals with deterministic demand paging algorithms noninformed about the continuation of the reference string /with so-called nonlookahead algorithms/.

In 1971 Aho, Denning and Ullman [1] proposed a method for the formal description of these deterministic, nonlookahead algorithms.

In his paper Belady [2] mentioned a stochastic algorithm -so-called random algorithm /RAN/. Recently a lot of other ones was proposed, for example in [3] and [4].

In the first part we extend Aho, Dennind, Ullman's terminology to the deterministic lookahead algorithms, and examine the relationship between two features /sequentiality and rationality/ of these algorithms.

In the second part is given a formal description of a subset of stochastic algorithms, which contains the known deterministic ones too. We extend the results of the first part to this set of algorithms.

PART I. DETERMINISTIC ALGORITHMS

1.1. Notations

We use Belady's mathematical model [2] of computers with two-level paged virtual memory. According to this model the computer consists of a central processor unit /CPU/, a main memory /MM/ and a backing store /BS/. MM and BS are divided to equal-size units, so-called pageframes. The programs are divided into paged corresponding to the pageframes. Processing a program we move its pages - if necessary - between the memory levels. Figure 1. shows the schema of such a computer.

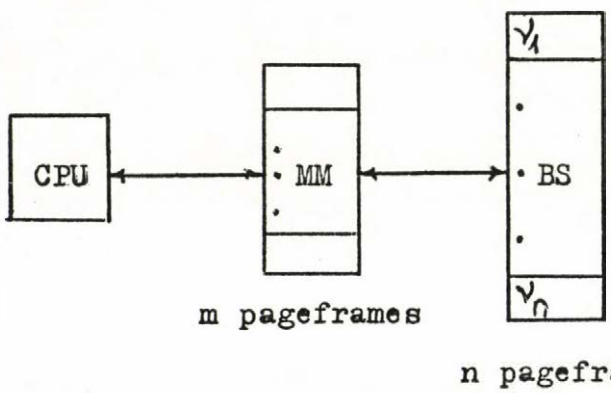


Fig. 1. Schema of a computer with two-level paged memory

We use the following notations.

m - the number of pageframes in MM $/1 \leq m < \infty /$;

n - the number of pageframes in BS $/m < n < \infty$

$N = \{\nu_1, \ldots, \nu_n\}$ - the set of pages, located in the pageframes of BS;

$\omega_T = r_1, \ldots, r_T$ $//r_t \in N,\ t=1, \ldots, T//$ - the reference string;

N^T - the set of all possible ω_T' ;

S_t - the state of MM at moment t /the set of the included pages $//S_t \subset N,\ 0 \leq |S| \leq m//$;

M^m - the set of all possible different S_t ;

$q_t = //q_p, q_f//_t$ - control state, which contains information about the past $//q_p//$ and the future $//q_f//$ of the reference string. We can remark, that q_p and q_f are both partially ordered subsets of N;

Q_p - the set of all q_p;

Q_f - the set of all q_f;

$Q = Q_p \times Q_f$ - the set of the control states;

S_o - initial memory state /often: $|S_o| = 0$/;

q_o - initial control state;

$g: M^m \times Q \times N \rightarrow M^m \times Q$ - mapping, which after each page request determines the new memory and control states;

$A = \langle N, M^m, Q, S_o, q_o, g \rangle$ sixtuple - the paging algorithm.

1.2. Definitions

1.2.1. Demand paging algorithms

In this paper we omit - if it is possible without the risk of misunderstanding - the indeces.

Further we discuss only the so-called demand paging

algorithms.

Definition 1 [1] A paging algorithm $A = \langle N, M^m, Q, S_o, q_o, g \rangle$ is demand paging algorithm, if its g is as follows:

$$g_A(S, q, \nu_i) = \begin{cases} (S, q'), & \text{if } \nu_i \in S \\ (S + \nu_i, q''), & \text{if } \nu_i \notin S \text{ and } |S| < m \\ (S + \nu_i - \nu_j, q'''), & \text{if } \nu_i \notin S \text{ and } |S| = m, \text{ where } \nu_j \notin S. \end{cases}$$

Let's take as example the well-known algorithm FIFO (first in - first out). FIFO is a demand paging algorithm, and replaces the page having spent the most time in MM. In the case of FIFO q_p contains the pages in MM in the order of their arrival:

$q_p = (y_1, \ldots, y_k)$, where $k \leq m$, and $y_i \in N$ $(i = 1, \ldots, k)$.

For FIFO q_f is not used. The mapping g_A is (if $|S| = m$) as follows:

$$g_{FIFO} = g_{FIFO}(S, q_p, x) = \begin{cases} (S, q_p), & \text{if } x \in S \\ (S + x - y_m, q_p'), & \text{if } x \notin S, \end{cases}$$

where $q_p' = (x, y_1, \ldots, y_{m-1})$.

Further on we often refer to the algorithm MIN, proposed by Belady [2]. This algorithm replaces that page being in the main memory which will be referenced at the latest.

Let's $\lambda_t = S_{t-1} \setminus S_t$ call the removed page during the processing of r_t ($\lambda_t \in N$ or $\lambda_t = \emptyset$).

Let $r_{t_1} = r_{t_2} = \nu_s$ ($t_1 \neq t_2$), $r_t \neq \nu_s$ ($t = t_1 + 1, \ldots,$

$t_2-1//$, then $\rho_t/\gamma_s/=t_2-t$ is the "forward distance" of page γ_s at moment t [5]. If $r_{t_1}=\gamma_s$ and $r_t\neq\gamma_s$ $/t=$ $=t_1+1, t_1+2, \ldots //$, then let $\rho_t/\gamma_s/=\infty$.

MIN also is a demand paging algorithm, according to which the removed page λ_t is such a page of S_t, for which holds

$$\forall\ \gamma_s\in S_t\quad \rho/\gamma_s/\leqslant\rho/\lambda_t/.$$

1.2.2. Processing costs

Definition 2 [1] The processing cost of a reference r_t in ω_T equals to

$$\delta_t=\delta/\omega_T, A, r_t/=\begin{cases}0, \text{ if } r_t\in S_{t-1}\\ 1, \text{ if } r_t\notin S_{t-1}.\end{cases}$$

Definition 3. The processing cost of the ω_T equals to

$$C=C/\omega_T, A/=\sum_{t=1}^{T}\delta/\omega_T, A, r_t/.$$

1.2.3. Sequential and rational paging algorithms

Definition 4. A paging algorithm is sequential [3], if

$$\forall\omega', \omega'', \omega'''\quad \sum_{t=1}^{|\omega'|}\delta/\omega'\omega'', A, r_t/=\sum_{t=1}^{|\omega''|}\delta/\omega'\omega''', A, r_t/.$$

According to this definition A is sequential, if the processing cost of the sequence ω_T till moment t depends

only of the first t references in ω_T.

For example MIN, and all the nonlookahead algorithms /FIFO, LRU, CLIMB etc./ are sequential.

Definition 5. A paging algorithm A is rational, if for every reference $\omega_T' = r_1' \ldots r_T'$ and $\omega_T'' = r_1'' \ldots r_T''$ the following conditions imply $\lambda_t'' = \nu_s$:

$$\exists t_1, \exists t_2 \quad \begin{aligned} &a/\ 1 \leq t_1 < t_2 \leq T \\ &b/\ \exists s \ \ \nu_s = r_{t_1}' = r_{t_1}'' \\ &c/\ \forall k \ \ /t_1+1 \leq k \leq t_2-1/ \ \ r_k' \neq \nu_s \\ &d/\ \exists t \ \ /t_1+1 \leq t \leq t_2-1/ \ \ \lambda_t' = \nu_s \\ &e/\ \forall j \ \ /1 \leq j \leq t_2/ \ \ r_j' = r_j'' \end{aligned}$$

According to this definition A is rational, if it does not use more information about the continuation of the reference sequence, than the algorithm MIN.

1.3. On the equivalence of rationality and sequentiality

In this section we examine the relation between the rationality and the sequentiality. The motivation for this is the following: the sequentiality can be well used in some examinations [3], but for a given paging algorithm it is easier to verify the rationality.

It is easy to show that these features are equivalent, if the algorithms don't use future information /q_f/.

The following two examples show that the equivalence isn't true in generality /for lookahead algorithms/, that is

the sequentiality doesn't imply the rationality and the rationality doesn't imply the sequentiality.

Example 1. Let the function g of A as follows:

$$g_A = \begin{cases} g_{MIN}, & \text{if } \gamma_1 \cup \gamma_2 \not\subseteq S \\ g_{LFU}, & \text{if } \gamma_1 \cup \gamma_2 \subseteq S. \end{cases}$$

It is easy to see that this algorithm is rational.

Let $m=3$, $\omega' = \gamma_4\ \gamma_3\ \gamma_2\ \gamma_2\ \gamma_1\ \gamma_1\ \gamma_5\ \gamma_4\ \gamma_3\ \gamma_1\ \gamma_2$

$\omega'' = \gamma_4\ \gamma_3\ \gamma_2\ \gamma_2\ \gamma_1\ \gamma_1\ \gamma_5\ \gamma_4\ \gamma_2\ \gamma_1\ \gamma_3$

Processing ω' and ω'' at $r_8 = \gamma_4$ the costs will be different $/\delta_8' = 0,\ \delta_8'' = 1/$, so the algorithm isn't sequential.

Example 2. Let L_t=true, if there exists such an s $/3 \leq s \leq n-2/$, for which the following conditions hold:

a/ $\gamma_1 \notin S_t$, $\gamma_2 \notin S_t$, $\gamma_{n-1} \in S_t$, $\gamma_n \in S_t$;

b/ $r_{t+1} = \gamma_1$; $r_{t+2} = \gamma_2$; $r_{t+3} = \gamma_1$; $r_{t+4} = \gamma_3$; $r_{t+5} = \gamma_4$; $r_{t+6} = \gamma_5$; ... $r_{t+n+1} = \gamma_n$; $r_{t+n+2} = \gamma_s$.

Let the function g_A as follows:

$$g_A/S_t, q, r_t/ = \begin{cases} /S_t - \gamma_{n-1} + r_t,\ q'/, & \text{if } L_t = \text{true and } r_{t+n+2} = \gamma_n \\ g_{MIN}, & \text{otherwise.} \end{cases}$$

This algorithm is not rational, because in some cases it needs more information $/r_{t+n+2}/$, than MIN.

However we can show that this algorithm is sequential. The algorithm A changes the memory state usually as MIN

does it. If A decides otherwise /replaces γ_{n-1} instead of γ_n/, then in the next step it correct the difference in the memory state /removing γ_n/ without difference in processing cost. Therefore A is sequential.

We can prove that under certain conditions a rational algorithm is sequential, because the following assertions are valid.

THEOREM 1. Every demand paging algorithm A having all the properties a, b, c, d is sequential:

a/ A is rational;

b/ $\forall q_p', q_p''$ $g/S,/q_p',q_f/,x/=g/S,/q_p'',q_f/,x/$;

c/ Let $S^* = \{r_t \mid r_t \in /S' \cap q_f/\}$, and $S' \cap S^* = S'' \cap S^*$, then $g_A/S',q,x/=g_A/S'',q,x/$;

d/ If for $\gamma_s \in S_t$ t_2 is as in Definition 5, and $\lessdot$ is the partial order in q_f, $\omega = r_1 \dots r_{t_2}$ and $\gamma_u \neq \gamma_s$, then for every ω', ω'' $\gamma_s \lessdot \gamma_u$ /in $\omega\omega'$/ implies $\gamma_s \lessdot \gamma_u$ /in $\omega\omega''$/.

For the simplicity we reformulate this assertion nonformally, formulate the other assertion only nonformally and give only heuristic proofs of these assertions.

Assertion 1. Every demand paging algorithm

a/ which is rational;

b/ which doesn't use q_p;

c/ for which the new q_f and the new S don't depend on the pages of S being absent in q_f;

d// for which q_f is limited by the rationality;

is sequential.

Proof of assertion 1. We use an indirect proof. Let's suppose that A is an algorithm with the above features but it isn't sequential: in this case there exist at least one ω_T, ω_1, ω_2, for which

$$\sum_{t=1}^{T} \delta /\omega_T\omega_1, A, r_t/ \neq \sum_{t=1}^{T} \delta /\omega_T\omega_2, A, r_t/.$$

So there is a time moment t $/0< t\leq T/$, that $\delta /\omega_T\omega_1,A,r_t/\neq \delta /\omega_T\omega_2, A, r_t/$. This inequality means, that at time t the algorithm remove a pagein one sequence but not in the other sequence. Hence the memory state S was different for a page which occures still in the equal sequence ω_T. Let's denote with t_1 the first moment, when the memory states become different in connection with a page still occuring in ω_T. Before this moment t_1 it holds $S_t'=S_t''$ //for all $t\leq t_1$/, and this step brings the first difference in S//. On the other hand $/q_f/_{t_1}'=/q_f/_{t_1}''$ for the page removed at t_1. But the function g_A uses only S and q_f to determine λ_t. So at time t_1 difference in S cann't occur. This contradiction can come only from $\sum_{t=1}^{T} \delta /\omega_T\omega_1,A,r_t/\neq \sum_{t=1}^{T} \delta /\omega_T\omega_2,A,r_t/$. Hence the algorithm A is sequential.

Assertion 2. Every demand paging algorithm

a/ which is rational;

b/ which doesn't forget the used future information;

c/ for which if the future information is available till rationality, then the function g is like as in point c/ of assertion 1;
is sequential.

Proof of assertion 2. The memory states /S/ during the processing of $\omega_T\omega_1$ and $\omega_T\omega_2$ can differ only if the algorithm "looked over" ω_T; in this case the algorithm knows the future till rationality for the pages still coming in the equal sequence. So g uses for this pages only q_f and S' according to b/. This doesn't cause difference in processing cost /see assertion 1./. On the other hand if no difference occures in memory state S, then the processing costs are also equal in the two sequences.

The majority of the deterministic paging algorithms is in the bounds of assertion 2. Assertion 1. is not so general, but it gives results for some special algorithms /e.g. future not forgetting algorithms/.

II. STOCHASTIC ALGORITHMS

A part of the paging algorithms - for example RAN [1], REF_a [3], MP_b [4], PP_c [6] - are stochastic ones. In this part, extending the method due to E. Gelenbe [7], we will describe a class of stochastic algorithms, which contains all the known by us deterministic and stochastic algorithms. We will extend the results of part 1. to this class of algorithms.

2.1. Notations

We leave the meanings of N, ω_T, N^T, n, m, S, M^m, q, Q without change and introduce the following new concepts:

$R = /\alpha_1, \ldots, \alpha_k/$ - stochastic memory state, where $k = \binom{n}{m}$ and α_i gives the probability of the event, that a given /the i-th/ set of m pages is in MM, and the sum of these α-s equals to 1;

R^* - the set of the stochastic memory states R;

$z = /\gamma_1, \ldots, \gamma_r/$ - the probability distribution of the control states, where $\gamma_i = P/q=q_i/$ and the sum of the γ-s equals to 1;

Z^* - the set of all z;

We have to extend the function g of 1.1. in the following way: $Q \times M^m \times N \to Z^* \times R^*$.

For deterministic algorithms this definition gives the old one.

In general case it is difficult to give the function g in a simple form. If the number of the control states is finite, then we can describe g with a matrix. Its size is $/|Q| \times |M^m| \times n/ \times /|Q| \times |M^m|/$.

If the algorithm is in the state /S,q/ and a page request for ν occures, then a row of the matrix gives the probabilities of the next state $/S', q'/$: the j-th element gives the transition-probability for the j-th state. The sum of the elements in a row equals to 1.

For practical purposes this matrix is too difficult.

Sometimes we use an other matrix: its size is $/|Q|x|M^m|xn/x|M^m|$ A row of this matrix gives only the memory state probability distribution. Beside this matrix we have to give a function which determines the new control state.

2.2. Modification of the definitions

Because the stochastic algorithms describe the memory states with some probabilities, we have to extend the definitions of processing costs. The processing cost of the reference r_t equals to $\delta/\omega_T, A, r_t/ = 1 - \beta_i$, where $r_t = \gamma_i$ and $\beta_i = P//\gamma_i \in S/$.

The processing cost of a T-length program realisation ω_T is the same as in 1.2., but we have to use the new definition of the reference cost.

We keep also the old definition of the sequentiality.

In the definition of the rationality we have to introduce a new condition beside the old ones, namely

$$P/\lambda_t' = \gamma_s/ = P/\lambda_t'' = \gamma_s/.$$

2.3. The relation between the rationality and sequentiality

The two assertions of the section 1.3. are also valid for stochastic replacement algorithms. We have to use the new definitions in the theorems.

In the proof of the assertion 1'. we have to use the new reference processing cost. The inequality $\delta/\omega_T, A, r_t/\neq$

$\neq \delta/\omega_T', A, r_t//$ means, that the probabilities $\beta = P/r_t \in S/$ were different in the two processings, hence the stochastic memory states were also different. The proof is otherwise unchanged.

The proof of the assertion 2. is essentially unchanged. We have to use - in accordance with the function g - the stochastic memory state R and the new processing cost as in the assertion 1.

This new assertion 2'. can be used for the majority of the deterministic and stochastic algorithms. For example MIN, REF_a, MP_b, PP_c satisfy its conditions. For these and some other algorithms it is easy to show the rationality, but to prove their sequentiality on an other way would be more difficult.

Acknowledgment. The authors are indebted to E. Z. Lubimski Professor of Moscow State University, who proposed to investigate this problem.

REFERENCES

1 Aho A. V., Denning P. J., Ullman J. D., Principles of optimal page replacement, J. of ACM, 18, No. 1. /1971/, 80-93.

2 Belady L. A., A study of replacement algorithms for a virtual storage computer, IBM Systems Journal, 5, No. 2. /1966/, 282-288.

3 Стоян Ю. А., Оценка эффективности алгоритмов замещения, Программирование, I, No. I. /1975/, 22-25.

4 Iványi A., Kátai I., On the speed of computers with paged and interleaved memory, MTA SZTAKI Közlemények, 18/1977, 105-118.

5 Coffman E. G., Denning P. J., Operating Systems Theory, Englewood Cliffs, Prentice Hall, 1973.

6 Стоян Ю. А., Результаты оценки эффективности оптимального алгоритма замещений, Программирование, I, No. 2. /1975/, 17-23.

7 Gelenbe E., A unified approach to the evaluation of a class of replacement algorithms, IEEE Trans. on Computers, 22, No. 6. /1973/, 611-618.

Összefoglaló

LAPOZÁSI ALGORITMUSOK TULAJDONSÁGAIRÓL

Bárdossy Dániel /VIDEOTON Fejlesztési Intézet/
Iványi Antal /Eötvös Loránd Tudományegyetem/

Az első részben a lapozási algoritmusok leírására eddig használt terminológiát kiterjesztjük előrenéző determinisztikus algoritmusokra, majd megvizsgáljuk, milyen összefüggés van ezen algoritmusok két tulajdonsága /sorosság és ésszerüség/ között.

A második részben megkisérreljük a sztochasztikus algoritmusok halmazának egy - valamennyi általunk ismert determinisztikus és valószinüségi algoritmust magában foglaló - részhalmazának formális leirását, és az első rész eredményeinek kiterjesztését erre az algoritmushalmazra.

Р е з ю м е

О НЕКОТОРЫХ СВОЙСТВАХ СТРАНИЧНЫХ АЛГОРИТМОВ

Д. Бардоши /Исследовательский Институт фирмы ВИДЕОТОН/
А. Ивани /Университет им. Лоранда Этвеша/

В первой части работы терминология, применённая для описания "слепых" страничных алгоритмов, распросраняется для детерминированных *некслепых* страничных алгоритмов. Рассматривается связь между двумя свойствами /последовательность и рациональность/ этих алгоритмов.

Во второй части сделана попытка на формальное описание такого подмножества вероятностных страничных алгоритмов, которое включает в себя все известные нам детерминированные и недетерминированные страничные алгоритмы. Результаты первой части распространены и на это подмножество страничных алгоритмов.

SOME RECENT DEVELOPMENTS IN DISK SYSTEMS UTILIZATION

György Győry
Computer and Automation Institute of the Hungarian
Academy of Sciences

I. Introduction Disk systems tend to serve more and more specified and sophisticated purposes. Thus their operating modes have to be more and more specific in order to catch up with the specific tasks to maintain.
The aim of this paper is to present two of the recent investigations concerning such adapted disk scheduling policies. One of these deals with specific dynamic allocation problems, in the case of which the sizes of the most frequently allocated areas are known. The other one examines the queneing properties of the rotational latencies of a disk system with identical starting addresses for all the stored records. This latter attempt aims the easier organization of a disk scheduling system at the expense of a less effective memory space allocation and thus persues aims in some sense contrary to those of the first.

II. A Fibonacci-like buddy system variation

A buddy system is a way for dynamic storage allocation with some large starting blocks of memory space. Should a demand arise for allocating a sufficiently small area, a free area of a suitably small size is looked for and if none is found, a larger one of the nearest size will be split to two buddies and the search restarted. Should the buddy of a free area be freed also, the buddies are reunited and if no further buddies become both free by this step, the united area will be recorded in a table suitable to its size.

Let the buddy sizes admitted in this procedure be $S_1, S_2 \ldots$ /in decreasing order/. Traditional buddy systems are characterized by splitting areas to two equal buddies and thus by the equation $S_k = 2 \cdot S_{k-1}$. Generalized Fibonacci-like systems split areas to buddies according to the equation

$$S_k = S_{k-1} + S_k - f(k) \qquad /1/.$$

In the case of the Fibonacci buddy system $f(k) \equiv 1$ and in most of the systems used in practice $f(k)$ is constant. The reason for this is simple: if the equality $f(k) = f(k-i) + i$ holds for any k, the starting address of the left buddy of a memory area with size $S_k - f(k)$ is ambignous and therefore must be stored either in the area itself or in a separate table /1/. Unless $f(k) \leq 1$ and is monotone nondecreasing, an other kind of anomaly may also arise. If the user's objective is to achieve a free memory area of a given size within the shortest time /i.e. performing as few splittings as possible/ the optimal way to accomplish this may not begin with splitting the smallest bigger area available. On the other hand, the splitting strategy may considerably influence the distribution of available areas of the different sizes. The original buddy system has the main advantage of easy and quick allocation and release algorithms /4/. On the other hand, it is not especially effective in utilizing memory space. The introduction of more sephisticated buddy systems obviosusly has the aim of better memory utilization. The cost of this new feature is more complicated algorithms for the allocation and release. Utilizing a generalized Fibonacci-like buddy system may have two major advantages if the function is suitably chosen: memory blocks of some required sizes may be obtained in a great number and margins of larger areas may be made to coincide with boundaries of physical blocks. When designing a generalized Fibonacci-like buddy system, the following two questions arise /which in the auther's opinion are worth further investigation/:

1. Which buddy function $f(k)$ should be used of those which provide us with the needed sizes of blocks with respect to the given distribution of demands over the sizes of blocks.

2. How should then the allocation procedure be chosen /with respect to the possibile different ways of splitting memory areas until one of the needed size is obtained/ to yield a reasonable compromise between short allocation times and keeping the size distribution of the free memory areas possibly close to the required one.
What makes the problems more complicated is that their solutions obviously depend on each other.

III. Rotational latencies of a disk system with uniform starting addresses

In this section rotational auxiliary memory devices /called disks for the sake of brevity/ will be considered with an operational system that assigns the same starting address to every record. Taking the time taken by one rotation of the disk for a time unit transfers of records may begin at instants some /integer number of/ time units after the first time the read /write head passes the initial address.
We shall assume that the record lengths have a known uniform distribution and denote by τ_n the probability that a record is transferred in n rotations. We assume also that the arrival process is Poisson with parameter λ. Let us denote by $M(t)$ the number of requests in queue and by $N(t)$ the number of disk revolutions having been taken by the current transfer at time t. The process $(M(t), N(t))$ /with $N(t)=-1$ if the queue is empty/ is not itself a Markov one, but it has an imbedded Markov chain at times t_i when the starting address passes the disk heads. The one-step transition probabilities of the latter are

$P((m,-1)|(m,-1)) = a_0$; $P((m,1)|(m,-1)) = a_m$ for $m \geq 1$;
$P((m,n)|(m-j,n-1)) = a_j c_{n-1}$ for $m \geq 1$, $n \geq 2$, $0 \leq j \leq m$;
$P((m,1)|(m-j+1,n)) = a_j (1-c_n)$ for $m \geq 1$, $n \geq 1$, $0 \leq j \leq m$;
$P((m,-1)|(1,n)) = a_0 (1-c_n)$

with $$c_n = P(x > T_n \mid x > T(n-1)) = \frac{1-F(T_n)}{1-F(T(n-1))}$$

and with zero probabilities for all the transitions not involved in the previous equations.
For the detailed proof of the following results the reader will be referred to /3/. First, the imbedded Markov chain can be shown to be aperiodic and irreducible. Then its ergodicity can be proved. Using this, with

$T = t_i - t_{i-1}$, $R(z) = \sum_{n=1}^{\infty} r_n z^n$,

$E\{w\} = \sum_{n=1}^{\infty} w r_w$ and $A(z) = e^{-\lambda T(1-z)}$

the probability of an empty queue

$$\gamma(0) = P(0,-1) = 1 - \lambda T E\{w\}$$

and the generating function of the state probabilities

$$P_n(z) = \sum_{m=1}^{\infty} \gamma(m,n) z^m = [1-F((n-1)T)]\gamma(0) z \left[\frac{A(z)-1}{z-R(A(z))}\right] A(z)^{n-1}$$

can be obtained, from which the state probabilities of the imbedded Markov chain $(M(t_i), N(t_i))$

follow. These results are generalizations of some corallaries of a theorem of Skinner /5/. A consequence of the previous result is that of Skinner's results as well.
By averaging the generating function of the state probabilities of the imbedded process at times $(t_i + n)$ with respect to n one gets the expected number of

requests in queue

$$E\{M\} = \lambda T\left(E\{l\} + \frac{1}{2}\right) + \frac{(\lambda T)^2}{2} \cdot \frac{E\{w^2\}}{(1 - \lambda T E\{w\})^2}$$

with l given by the proportion of the length of a transfer to the time unit $(t_i - t_{i-1})$.

Moreover,

$$E\{w^2\} = \sum_{w=1}^{\infty} w^2 r_w \quad .$$

In the possession of these results the effectivity of the disk scheduling policy with uniform starting addresses can be considered. The primary one being the simplicity /of the addressing policy/ our question is if it is worth the deterioration of the mean response time and the fragmentation of the disk space. From the latter point of view this policy at first sight occurs opposite to that of the generalized buddy system. It is true, that their aims are totally different, but without further knowledge of the record size distributions no condusion can be drawn. An approximately uniform distribution of the record sizes is disadvantageous for both, but it is more for the buddy system. On the other hand, if all the records are of the same length, the buddy system cannot be conveniently used either. In both cases the mean response time of the uniform starting address policy is next to that of the FIFO scheduling in the case of uniformly distributed starting addresses and thus for busy states of the system considerably worse than the mean response time of the Shortest-Latency-Time-First policy at uniform starting addresses. The estimations have been derived from the results of /3/ and the simulation results of /2/.

References

/1/ Burton, W.
A Buddy System Variation for Disk Storage Allocation
Comm. ACM 19/7 /1976/ 416-417

/2/ Fuller, S. H.
Random Arrivals and MTPT Disk Scheduling Disciplines
Tech. Rep. 29, Digital Systems Lab. Stanford Univ.,
Stanford, Calif. 1972

/3/ Gelenbe, E.; Lenfant, J.; Potier, D.
Response Time of a Fixed Head Disk to Transfers
of Variable Length
SIAM J. of Comput, 4/4 /1975/ 461-473

/4/ Knuth, D. E.
The Art of Computer Programming, Vol. 1
Addison-Wesley, Reading, Mass. 1968, 442-445.

/5/ Skinner, C. E.
A Priority Queueing System with Server-Walking Type
Operations Res. 15 /1967/, 278-285.

ÖSSZEFOGLALÓ

Diszkrendszerek használatának újabb módjairól

Győry György

A dolgozat korábbi cikkek alapján két allokálási stratégiát ismertet, és azok effektivitását hasonlitja össze az allokálandó területméretek eloszlása szerint. Az egyik stratégiánál a rekordok kezdőcime rögzitett, a másik az ismert "buddy system" általánositása.

РЕЗЮМЕ

О новых методах исползования дисковых систем

Дьёрдь Дьёри

Настоящая работа на основе раннее опубликованных работ описывает две стратегии аллокации и сравнивает их ефффективность по распределению памяти.

При первой стратегии начальные адреса записей фиксированы. Вторая стратегия является обобщением известной "buddy system".

Coordination of dynamic systems of processes

Előd Knuth
Computer and Automation Institute Hungarian
Academy of Sciences

1. Dynamic case

means systems consisting of permanently variable number of cooperating processes. The control structure of a dynamic system may be described by static /fixed/ models in some of the cases. However, when this way can not be chosen, the realization and the verification of the synchronization rules becomes much more difficult.

This paper introduces a reasonable general rule which can be used to build algorithms in a wide class of applications and makes it possible to reduce the problem to those used in the static case.

Let Ψ be the set of synchronization rules controlling the system. Since the system is dynamic, the set Ψ is permanently changing during execution time.

The separation principle we propose is the division of Ψ into disjoint classes such that

1. There is no operation in the system synchronized by rules belonging to different classes;

2. A set of rules contituting a class can be generated /and later released/ always together.

If we can comply with this principle in programming, we shall be able to verify a lot of properties from the static viewpoint, and the dynamic nature will be reduced to the interconnections among classes. The principle described is illustrated by the example of

2. Telecommunication processes

which usually contain two types of critical actions:

D - declaring a demand,
/placing a notice into a queue/;

C - passing a checkpoint
/testing whether certain demand is served/.

Demands declared are served /action **S**/ by a central monitor routine which works according to certain scheduling strategie and which is driven by the information contained in queues.

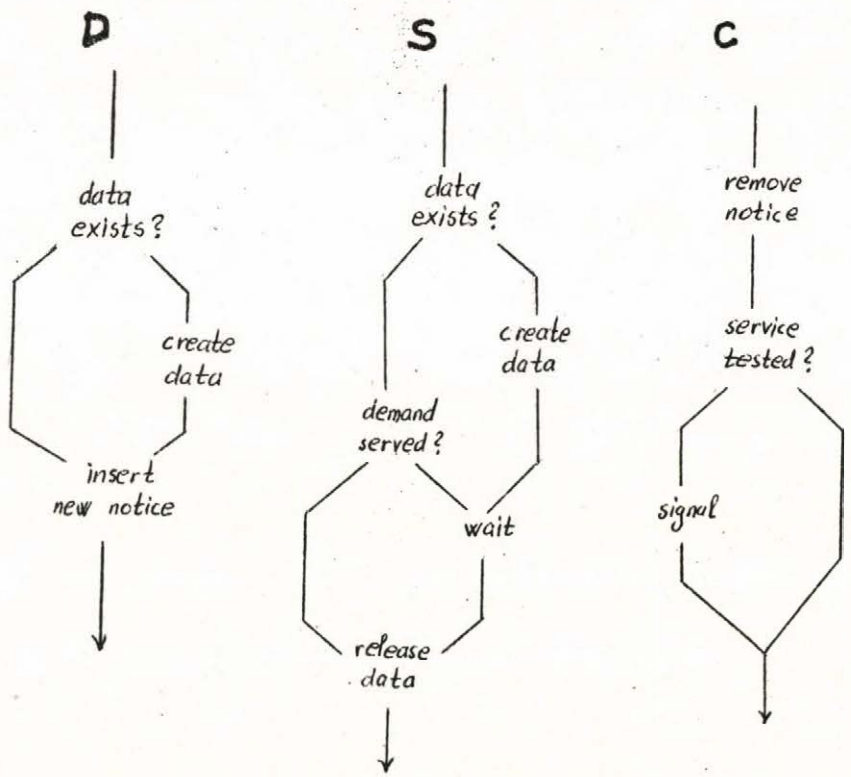
Processes can create a collection of subprocesses and even subprocesses can do so. Actions **D** and **C** can be contained by different subprocesses therefore we have no information on which of them occure first.

Demands declared by processes can be of different types i.e. they can belong to different queues. However, at a time more than one demand can be declared for the same queue and they may be satisfied at different time points. Moreover, the number of processes and subprocesses working together is unknown too.

This complicated cooperation can be made well structured by making the operation rules disjoint. Consider the sets of logically connected triplets of critical actions: **/D, S, C/.** For each of the triplets the following requirements must be fulfilled:

1/ **S** can not precede **D.**
2/ **C** must delay the executing process until the demand tested is both declared and served.
3/ If a process is waiting for a certain service, its execution must awake the process.
4/ A process testing a demand already served must continue its progress.

Suppose we generate an istance of a data structure registrating the status of the cooperation rule for each triplets of logically connected actions. Now, for example the following algorithms can be used in a quasi-parallel environment:



As the cooperation in one triplet is a fixed /static/ rule, it is very easy to verify all the four requirements we prescribed for example by using the usual assertion method /see e.g. [1]/. By other side, the disjointness of the triplets guarantees the correct coordination of the dynamic system as a whole.

Reference

1. Owicki, S., Gries, D. An axiomatic proof technique for parallel programs. Acta Informatica 6, 319-340 /1976/.

Folyamatok dinamikus rendszerének koordinálásáról

E. Knuth
MTA Számitástechnikai és Automatizálási Kutató Intézete

A dolgozat konkurrens folyamatok programozásával foglalkozik abban az esetben, amikor a koordinálandó folyamatok száma változó.

Координация динамических систем процессов

Э. Кнут
Исследовательский Институт Вычислительной Техники и Автоматизации Ван

В статье рассматривается случай программирования конкурентных процессов, составленных из переменного числа компонентов.

ON RESOURCE ALLOCATION POLICIES IN UNIPROCESSOR SYSTEMS WITH NONPREEMPTIBLE RESOURCES

Wojciech Cellary

Institute of Control Engineering

Technical University of Poznan

Poznan , Poland

ABSTRACT. The problem of nonpreemptible resource allocation in uniprocessor systems to minimize schedule length is considered. The solution of the performance failures problems such as deadlock and determinacy is taken into account. The optimal and heuristic methods are presented and compared with classical deadlock avoidance method.

1. INTRODUCTION

In this paper we consider the problem of optimal resource allocation in computer systems. One can distinguish two aspects in the general case of such a problem :

- firstly - the optimization of a given performance measure , and
- secondly - the solution of the performance failures problem.

Performance failures in the system here mean :

- first, deadlock which is a system state in which the progress of some tasks is blocked indefinitely because each task holds nonpreemptible resources that must be acquired by others in order to proceed , and
- second, the determinacy of the set of tasks; the set is non-determinate if the results produced by independent tasks depend on the order in which these tasks are executed.

Until today, these two aspects of optimal resource allocation were considered separately. There are a number of algorithms which deal with scheduling tasks on processors, but without regard to other resources. On the other hand, known approaches to solving the problems of performance failures in the system do not explicitly take into account any performance measure. In this paper we will present an approach to the solution of general problem of optimal resource allocation, i.e., such an approach to the solution of the performance failures problem, which allows for the explicit formulation of the optimization problem. We will consider the minimization of schedule length as a system performance measure.

In Section 2 we formulate the problem precisely and present approaches to the solution of the performance failures problems. Sections 3 and 4 deal respectively with the optimal and heuristic approaches to the solution of the considered problem. Section 5 contains some computational results and Section 6, final conclusions.

2. PROBLEM FORMULATION

As was mentioned, our goal is to solve the performance failures problem together with the minimization of schedule length as a system performance measure.

First,let us consider approaches to the solution of the deadlock problem. As is known, there are three general approaches to dealing with deadlocks: detection and recovery, prevention, and avoidance [1]. We disregard the detection method since, in general, recovery is not allowed. We will use a prevention method, because the application of the avoidance method does not allow for the explicit formulation of the optimization problems for any system performance measure. However, the avoidance method improves resource usage as a consequence of its formulation and, as a result, tends to minimize schedule length. Therefore, in Section 5, we will compare the presented approach with the classical avoidance method given in [4].

Let us pass now to the determinacy problem. As is known, this problem can be solved simply by introducing the additional, proper precedence constraints between tasks [1]. Thus, considering sets of dependent tasks, it is possible to solve the determinacy problem.

In our approach, we will assume that every task is split into the maximum number of operations /see Fig. 1/, where an operation is the basic system unit which can request resources. We assume that an operation is characterized by its performance time, and its resource request vector. Moreover, all resources held by an operation are released

after its completion. This assumption allows for deadlock prevention because no operation holds resources while waiting for others to release resources.

Let us now explain the problem of requesting a central processor. We will use the idea of "processor sharing". This means that an operation requests some percentage of processor power only. The operation performance time, multiplied by the percentage of processor power, is equal to the processor work time during operation performance. The idea of processor sharing allows one to consider parallel processing in a uni-processor system.

Sum-marizing, we will consider the minimization of schedule length for a set of dependent operations, each operation characterized by its performance time and resource request vector.

3. OPTIMAL APPROACH

It can be proved that the problem formulated here is NP-complete [3]. This means that polynomial in time, optimal methods for solving it probably do not exist. Thus, it is necessary to look for simple suboptimal policies which could be used in operating systems. However, enumerative, optimal methods are also needed at least for the evaluation of the suboptimal policies. Let us start with these methods.

Note, that in Section 2 we have in fact formulated a multiple constrained resource, project scheduling problem for which there exists a rich bibliography. For solving the problem we propose the method given by Patterson and Huber [5] since

it has valuable computational properties. It uses the zero--one approach together with the minimum bounding technique which consists in establishing a lower bound on the resulting zero-one programming problem to see if it has feasible solution. When the solution of the zero-one problem is not feasible, schedule length is increased by one time unit and the feasibility examination is repeated.

Let us note that in this method the zero-one opproach is used only for the examination of the solution feasibility. The advantage of this method over using zero-one programming without employing any form of bounding [6] , is the great reduction in the computational time needed to reach the optimal solution.

Let us introduce now some definitions to be used in this section and formulate the zero-one problem.

We will consider time split into periods /t=1,2,...T; T equals the last period in the scheduling horizon/.

Every operation j /j=1,2,...,N/ will be characterized by the following :

τ_j - performance time /in integer time periods/;

e_j - the erliest possible period in which operation j could be completed /by virtue of operation precedence constraints/;

l_j - the latest possible period in which operation j could be completed /by virtue of operation precedence constraints and due date/;

r_{jk} - operation j requirements of units of resource k /k=1,2,...,K/;

$$x_{jt} = \begin{cases} 1 & \text{if operation } j \text{ is completed in period } t, \\ 0 & \text{otherwise,} \end{cases}$$

$$/x_{jt} \equiv 0 \text{ for } t < e_j \text{ and } t > l_j/.$$

The set of operations is characterized by the following parameters :

CP - the critical path length /in time periods/;

e - the earliest possible period by which the set of operations could be completed /by virtue of precedence constraints/;

$$x_t = \begin{cases} 1 & \text{in period } t \text{ if all operations have been} \\ & \text{in or before period } t; \\ 0 & \text{otherwise,} \end{cases}$$

$$/x_t \equiv 0 \text{ for } t < e/.$$

Moreover, let R_k denote the amount of resource k available in the system /assumed constant/.

The zero-one formulation of the problem is as follows.

Maximize

$$\sum_{t=e}^{T} x_t \qquad /1/$$

Subject to:

$$\sum_{t=e_j}^{l_j} x_{jt} = 1 \quad , \quad j=1,2,\ldots,N \; ; \qquad /2/$$

$$x_t \leqslant (1/N) \sum_{j=1}^{N} \sum_{q=e_j}^{t-1} x_{jq} \quad , \quad t=e,e+1,\ldots,T; \qquad /3/$$

$$\sum_{t=e_m}^{l_m} t x_{mt} + \tau_n \leqslant \sum_{t=e_n}^{l_n} t x_{nt} \quad , \quad m \ll n \; ; \qquad /4/$$

$$\sum_{j=1}^{N} \sum_{q=t}^{t+\tau_j-1} r_{jk} x_{jq} \leqslant R_k \quad , \quad \begin{matrix} t=1,2,\ldots,T \; ; \\ k=1,2,\ldots,K \end{matrix} \qquad /5/$$

The meaning of the above formulation is as follows. Minimization of schedule length is equivalent to maximization of the number of time periods remaining after completion of the set of operations /1/.

Equations /2/ define variables x_{jt} and guarantee that every operation will be completed. Inequalities /3/ define variables x_t and guarantee the completion of the whole set of operations. Inequalities /4/ correspond to the precedence constraints / $\ll$ denotes "immediately precedes"/, and /5/ - - to the resource constraints.

The minimum bounding algorithm may now be presented.

1°. Let T equal the smallest integer greater than or equal to $\max \left\{ CP, \max_k \left[(1/R_k) \sum_{j=1}^{N} r_{jk} \tau_j \right] \right\}$.

2°. Set up a zero-one integer programming problem using /1/ - /5/ with T as the maximum finish period. Call this problem P .

3°. If P has a feasible solution stop the algorithm, otherwise, replace T by T+1 and repeat from step 2°.

In step 1^{o} of this algorithm we determine the lower bound on the schedule length, which is either the critical path length - CP , or the maximum resource work content for a particular resource divided by the quantity of that resource available in the system. After termination of the algorithm, the schedule with the shortest length will have been found, since shorter length schedules are certainly infeasible.

4. HEURISTIC APPROACH

There are known a number of heuristic approaches to the solution of the multiple constrained resource, project scheduling problem [2] . The shared idea of these methods is as follows. We construct, step by step, a schedule increasing the size of a partial schedule by an operation which satisfies precedence constraints. From among operations which can increase the size of a given partial schedule we choose the one with the highest priority. Priorities are associated with operations in accordance with a given priority rule. The main difference between the considered methods simply consists in the choice of the priority rule. From the comparison provided by Cooper [2] , it follows that there are six priority rules which produce the best results. From among them, after computational experimentation for typical sets of tasks, we have chosen the rule which uses the total time of successors of an operation as its priority.

Let us formulate this method more precisely.

Let S_j and S_j^{-1} denote sets of predecessors and successors of an operation j, respectively. Let a partial schedule be

a specification of start times for a subset X of a set of operations, such that if $j \in X$ then so $i \in X$ for all $i \in S_j$, and such that the resource constraints are not violated. To increase the size of a partial schedule we have to determine a start times satisfying the resource constraints for some operation j; this operation cannot be in X , but all its predecessors must be in X. The set of operations which satisfy this requirement will be denoted by Q_X , i.e., $Q_X = \{ j : j \notin X, i \in X, \text{for all } i \in S_j \}$.

The algorithm can now be presented. It consists of the following steps.

1°. Enter, with an empty partial schedule /$X = \emptyset$ /.

2°. Create Q_X . If $Q_X = \emptyset$ then stop the algorithm.

3°. Calculate $p_j = \tau_j + \sum_{i \in S_j^{-1}} \tau_i$ for all $j \in Q_X$.

4°. Choose operation j* from Q_X with the maximum p_{j*} .

5°. Schedule operation j* /find the earliest possible start time, such that the resource and precedence constraints are not violated/ and return to step 2°.

5. COMPUTATIONAL RESULTS

The proposed heuristic approach was compared with the classical avoidance approach [4]. 70 typical sets of tasks were scheduled both by the proposed algorithm and the avoidance method. In 74% of the cases the proposed algorithm gave shorter schedule lengths, by 15% on average.

Especially valuable results were obtained when :

1. Tasks could be split into a great number of operations.
2. The precedence constraints were relatively stronger then resource constraints.

The first postulate is very often fulfilled. Typically, a task need some resources for a long initial period /spooling of input files/ and for a long terminal period /spooling of output files/, so that it can be split into separate operations.

The second postulate is fulfilled if many tasks use a common data base and the solution of the determinacy problem requires the imposition of many additional precedence constraints.

Moreover, let us note that the ratio of the overhead involved in the proposed algorithm to the overhead involved in the avoidance method is on average of the order $1/(1.5*N)$ since the avoidance method is a dynamic rule, and thus must be performed for every resource request.

6. FINAL REMARKS

The proposed approach allows one to consider explicitly schedule length minimization in systems with nonopreemptible resources. The system performance failures problems, such as deadlocks and determinacy, are solved, and moreover the proposed solution of the deadlock problem aims to increase resource utilization.

The comparison of the proposed method with the classical avoidance approach shows the advantage of this method, especially when the determinacy problem is set up, and the

task performance type is reading-processing-writing.

REFERENCES

1. Coffman, E.G. Jr., P.J. Denning : Operating Systems Theory, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1973,

2. Cooper, D.F.: Heuristics for scheduling resource-constrained projects: an experimental investigation, Manag. Sci. 22, 11, 1976.

3. Garey, M.R., D.S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, SIAM J. Comput. 4, 4, 1975.

4. Habermann, A.N.; Prevention of system deadlock, Comm. ACM 12, 7, 1969.

5. Patterson, J.H., W.D. Huber : A horizon-varying, zero-one approach to project scheduling, Manag. Sci. 20, 6, 1974.

6. Pritsker, A.A.A., L.J. Watters, P.M. Wolfe; Multiproject scheduling with limited resources a zero-one programming approach, Manag. Sci. 16, 1, 1969.

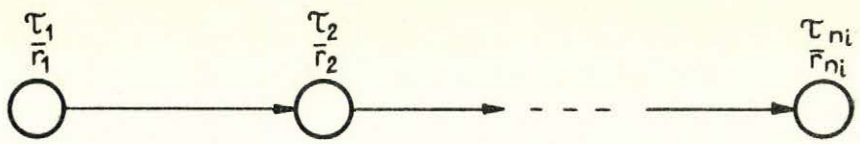


Fig. 1. Task structure

Erőforrás lekötési politikák a nempreemptiv esetben

Cellary W.

A dolgozat a minimális ütemezés determinisztikus vizsgálatával foglalkozik.

Р Е З Ю М Е

Политика завязывания ресурсов

В. Целлари

В статье рассматривается проблема минимального расписания.

KVÁZI PARALLEL TASKRENDSZEREK

Gáspár A., Visontay Gy., Csáki P.:
MTA Számitástechnikai és Automatizálási
Kutató Intézete

Összetett számitógépes feladatokat rendszerint önálló programok - ugynevezett taskok - programozott együttmüködésével oldunk meg. Másszóval lehetővé tesszük, hogy a taskrendszer elemei megszakitsák müködésüket, átadják egymásnak a vezérlést, majd folytassák futásukat a megszakitási ponttól.

1. A task állapotok.

Egy taskrendszerben a taskok öt állapotát célszerü megkülönböztetni:

- Uj task az ugynevezett taskkönyvtárból léphet be a taskrendszerbe. Ilyenkor egy uj taskpéldány keletkezik. A taskkönyvtár önálló futásra képes taskokat tartalmaz, melyek prototipus állapotban vannak /taskkönyvtár lehet például az operációs rendszer főkönyvtára, felhasználói könyvtár vagy egy file/.
- Az együttmüködő taskok közül aktivnak nevezzük azt, amelyiknél éppen a vezérlés van. Ilyen mindig legfeljebb egy van a rendszerben.
- Várakozónak nevezzük azokat a taskokat, amelyek a központi memóriában helyezkednek el, de nem aktivak. Aktivvá válhatnak viszont, ha vezérlést kapnak az aktiv tasktól.
- Passziv állapotunak nevezzük azokat a taskokat, amelyek saját futásukat félbeszakitották és nincsenek a központi memóriában. Ezek tárolására disken egy ugynevezett RUN TIME LIBRARY-t létesithetünk.
- Terminált állapotunak nevezzük azt a taskot, melynek futása véglegesen megszakadt, az általa lefoglalt központi memória pedig felszabadult.

2. A taskmüveletek.

A taskrendszer első taskját JØB CØNTRØL utasitással aktivizáljuk. Az aktiv task végrehajthat ugynevezett taskmüveleteket, melyek azonositói a következők: TASKCALL, TASKINTERRUPT, TASKTERMINATE, TASKPASSIVATE, TASKACTIVATE. Minden taskmüvelet két taskra hat. Egyikük mindig az aktiv task. Egy-egy taskmüvelet hatására:

- az aktiv task az előirt állapotba kerül és a másik task válik aktivvá
- létrejön /TASKCALL, TASKACTIVATE esetén/ illetve megszünik /TASKINTERRUPT, TASKTERMINATE, TASKPASSIVATE esetén/ a két task között az ugynevezett hivási kapcsolat.

Egy ujonnan létrejött hivási kapcsolatban a taskmüvelet előtti aktiv taskot hivónak, a taskmüvelet utáni aktiv taskot pedig hivottnak nevezzük. A hivási kapcsolatot létrehozó taskmüveletek hatására a hivó task várakozó állapotba kerül, mig egy hivási kapcsolatot megszüntető taskmüvelet hatására ismét a hivó task válik aktivvá. Vagyis TASKINTERRUPT, TASKTERMINATE, TASKPASSIVATE esetén az aktiv task hivója kap ismét vezérlést.

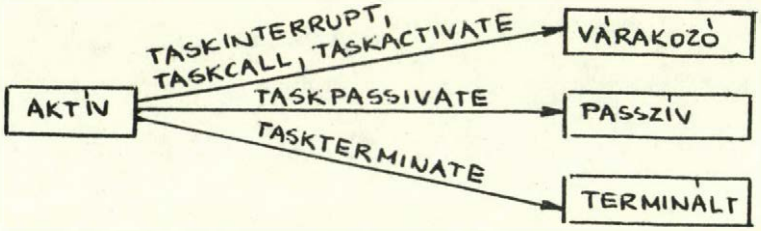
Hivási láncnak nevezzük a taskok egy olyan $T_1, T_2, \ldots T_n$ sorozatát, ahol

- a T_1 JØB CØNTRØL utasitással aktivizált
- a T_n aktiv
- a T_{i+1} hivója T_i $(1 \leq i \leq n-1)$

A TASKACTIVATE illegális taskmüvelet a hivási lánc várakozó állapotú elemeinek aktivizálására.

ÁLLAPOTVÁLTOZÁSOK A TASKMÜVELETEK HATÁSÁRA

A müvelet előtt aktiv task lehetséges állapotváltozásai



A müvelet hatására aktivvá váló task állapotváltozása

3. A taskrendszerek kezelését támogató eljárások.

Megterveztünk és implementáltunk egy eljáráscsomagot, melybe a következő eljárások tartoznak:

a. A fent definiált taskmüveleteket megvalósitó TASKCALL, TASKINTERRUPT, TASKTERMINATE, TASKPASSIVATE és TASKACTIVATE eljárások.

b. A TASKSAVE eljárás, mely segitségével az aktiv task másolatot készithet magáról a RUN TIME LIBRARY-be; a TASKRESTORE eljárás, mely segitségével egy korábban készült másolattal felülirhatja magát.

c. A RUN TIME LIBRARY létrehozását és karbantartását biztositó RTLGENERATE, RTLCOMPRESS, RTLREPORT és TASKDELETE eljárások.

d. A taskok közötti információátadást a RUN TIME LIBRARY kommunikációs mezőin keresztül megvalósitó PUTFIELD és GETFIELD eljárások.

Az eljáráscsomagot CDC 3300 SIMULA/MASTER számára implementáltuk, lásd [3].

A jelen anyagban azonban csak az RTLGENERATE, TASKPASSIVATE, TASKACTIVATE, TASKCALL, PUTFIELD és GETFIELD eljárásokkal foglalkozunk kissé részletesebben.

A RUN TIME LIBRARY használata lehetővé teszi, hogy a központi memóriában helyet szabaditsunk fel, és például egyidőben csak a futó /aktiv/ task és az őt hivó /várakozó/ task legyen a központi memóriában.

A fenti fogalmak és eszközök segitségével bonyolult taskrendszerek /például nagyméretü szimulációs feladatok/ kezeléséhez lehetőségünk van rendszermonitort irni. E taskrendszerek éppen passziv elemei a RUN TIME LIBRARY-ben helyezkednek el. A taskrendszer strukturájának, állapotának modelljét ilyenkor a monitor tartalmazhatja és a taskrendszer egy-egy eleme a monitorban tárolt modellen történő elemzés eredményeképpen aktivizálódhat.

E dolgozat végső célja egy konkrét, a SIMULA 67 programozási nyelv kvázi parallel folyamatkezelésének elvén alapuló rendszermonitor modell ismertetése.

4. A RUN TIME LIBRARY fogalma.

A RUN TIME LIBRARY egy az RTLGENERATE eljárás által létrehozott disk-file, melynek tartalma:

- Taskok passziv állapotban /lásd TASKPASSIVATE eljárás/, a passziválási pontjuktól ujraaktivizálható formában /lásd TASKACTIVATE eljárás/.
- Taskok közötti kommunikációra szolgáló, standard méretü, programozható nevü, változtatható tartalmu kommunikációs mezők /lásd PUTFIELD, GETFIELD eljárások/.

A RUN TIME LIBRARY kezelését teljes mértékben a felhasználó programozza, mig a fejlettebb operációs rendszerekben a hasonló szerepet betöltő eszközök nagyrésze csak az operációs rendszerek számára érhető el. A RUN TIME LIBRARY egy uj könyvtárfogalom, melyet hosszú idejü számitások és nagy memóriaigényü taskrendszerek számára konstruáltunk. A RUN TIME LIBRARY és a kezelésére kialakitott eljáráskészlet egyéb felhasználási területe ismeretlen. Hatékony felhasználása érdekében meg kell terveznünk a ráépülő különféle szervezettségü taskrendszerek modelljeit /lásd taskrendszerek monitorai/.

5. A fontosabb eljárások részletesebb ismertetése.

```
PROCEDURE RTLGENERATE (LIBSIZE);
    INTEGER LIBSIZE;...
```

Funkció: A RUN TIME LIBRARY kezdőállapotának generálása.

```
PROCEDURE TASKCALL (POINTNAME,TASKNAME,LIBNAME,MESSAGE);
    VALUE POINTNAME, TASKNAME,LIBNAME,MESSAGE;
    TEXT POINTNAME,TASKNAME, LIBNAME,MESSAGE;...
```

Funkció: Taskhivás taskkönyvtárból.

PROCEDURE TASKPASSIVATE(POINTNAME,STATENAME);

VALUE POINTNAME,STATENAME;

TEXT POINTNAME,STATENAME;...

Funkció: A futó task pillanatnyi állapotáról másolat készitése a RUN TIME LIBRARY-be majd a task terminálása.

PROCEDURE TASKACTIVATE(POINTNAME,STATENAME);

VALUE POINTNAME,STATENAME;

TEXT POINTNAME,STATENAME;...

Funkció: Passziv task visszahivása a RUN TIME LIBRARY-ből.

PROCEDURE PUT FIELD(FIELDNAME,FIELDTEXT);

VALUE FIELDNAME, FIELDTEXT;

TEXT FIELDNAME, FIELDTEXT;...

Funkció: Kommunikációs mező létrehozása/módositása/megszüntetése a RUN TIME LIBRARY-ben.

TEXT PROCEDURE GETFIELD (FIELDNAME);

VALUE FIELDNAME;

TEXT FIELDNAME;...

Funkció: RUN TIME LIBRARY-beli mező tartalmának szolgáltatása.

Paraméterek:

POINTNAME - Tetszőleges, a hivási/passziválási/aktiválási pont helyének azonositására alkalmas szöveg.

TASKNAME - A hivott task neve.

LIBNAME - A hivott taskot tartalmazó könyvtár azonositója.

MESSAGE - A hivott task számára átadandó paraméter string.

LIBSIZE - A RUN TIME LIBRARY méretét meghatározó számérték.

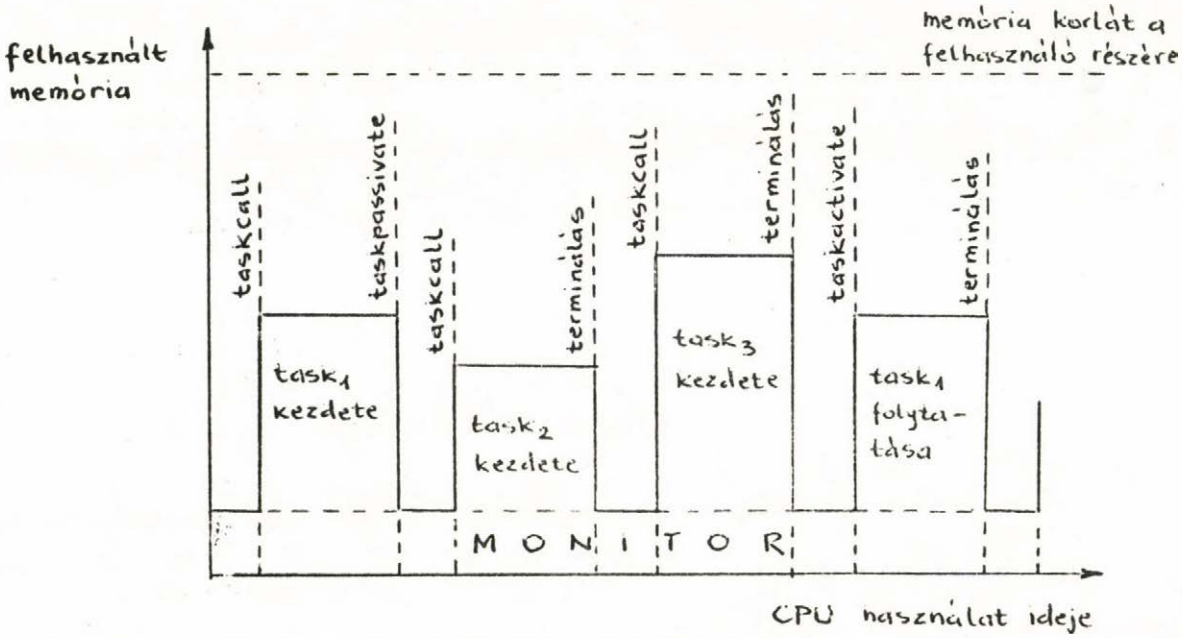
STATENAME - A RUN TIME LIBRARY-be készitendő, vagy az onnan visszahivandó másolat neve.

FIELDNAME - A létrehozandó/módositandó/megszüntetendő vagy a keresett kommunikációs mező neve a RUN TIME LIBRARY-ben.

FIELDTEXT - A kommunikációs mező tartalma.

6. Monitorral müködő taskrendszerek szervezettsége.

Taskrendszerek futtatását monitorral célszerü szervezni, ami azt jelenti, hogy a taskrendszer elemeit rendre a monitor aktivizálja és az aktiv taskpéldány passziválása, illetve terminálása esetén a vezérlést a monitor kapja vissza. Ilyen szervezés esetén a központi memória foglaltságának folyamatát az alábbi ábra szemlélteti:



Az ábrán az aktiv taskot satirozás jelzi. Ebben az esetben a taskrendszert a következőképpen szervezhetjük: A monitorban tároljuk a rendszer egy modelljét. Egy taskpéldány monitorbeli modelljét drivernek nevezzük. A driver egy monitoron belüli adatstruktura, melynek attributumai a hozzárendelt taskpéldány állapotára vonatkozó információkat tartalmazzák. A driverek által alkotott struktura /például fa/ tükrözi a taskrendszer pillanatnyi strukturáját, állapotát. A továbbiakban a taskokat nagybetükkel, driverüket pedig a megfelelő betü felülhuzásával jelöljük és megforditva. Például az X task drivere $\overline{X}$, és megforditva az $\overline{X}$ driver az X taskot reprezentálja. A későbbi ábrákon kizárólag drivereket ábrázolunk a kör szimbólum segitségével.

A taskrendszerekben kétfajta vezérlésátadási módot különböztetünk meg.

Decentralizált vezérlésátadásról akkor beszélünk, ha a taskrendszer éppen aktiv eleme a saját állapotáról szóló jelentésen kivül beleszól még abba is, hogy passziválása illetve terminálása után a monitor melyik taskot aktivizálja.

Centralizált viszont a vezérlésátadás akkor, ha az aktiv taskpéldány kizárólag állapotjelentést ad. Ilyenkor a monitornak kell eldönteni, hogy melyik taskot kell következőnek aktivizálni.

Mindkét tipusu vezérlésátadás esetén a taskpéldányok a monitorral a következőképpen kommunikálhatnak:

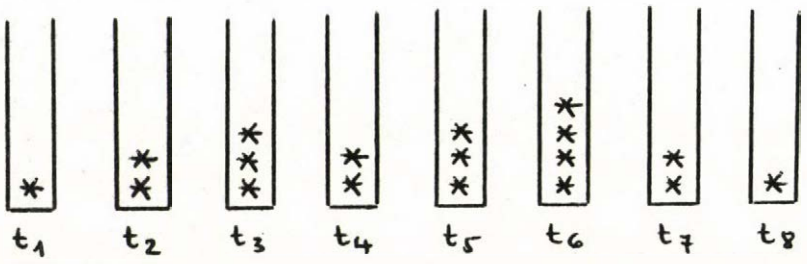
Egy-egy taskpéldány az állapotáról szóló jelentést és a vezérlésátadást előiró parancsát /ha van ilyen/ eljáráshivásokon keresztül

helyezi el a RUN TIME LIBRARY kommunikációs mezőin. Az eljárás feladata gondoskodni az állapotjelentés és a parancs összhangjáról. Vezérlésátadáskor a monitor a neki szánt állapotjelentés és parancs értelmezése, majd a driverek által alkotott struktúrán való elemzés után választja ki a következőnek meghivandó taskot.

7. ALGOL 6o szerkezetü taskrendszerek.

Az ALGOL tipusu nyelvekben a program épitőelemei a blokkok, a taskrendszerek épitőelemei pedig a taskok. E pont célja megvizsgálni azt, hogy az ALGOL tipusu nyelvek jól bevált blokkstruktúrái megfelelők-e taskrendszerek szerkezeteként is.

A következő ábrán egy ALGOL 6o program futásának folyamatát szemléltetjük. Az ábrán a blokkokat csillaggal jelöljük, a program futásának pillanatnyi állapotát pedig egy blokkokat tartalmazó verem ábrázolja. A verem alján a legkülső, a verem tetején pedig az éppen aktiv legbelső blokkot reprezentáló csillag foglal helyet.



Az ábrán t_i-k az ALGOL 6o program blokkmüveletei végrehajtásának időpontjait jelentik. Az ábrázolt ALGOL-folyamatban a blokkmüveletek a következők voltak:

belépés blokkba t_1, t_2, t_3, t_5, t_6-ban

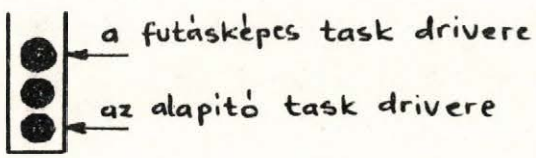
kilépés blokkból t_4, t_7, t_8-ban.

Az ALGOL 60-ban a blokkból kilépésnek két változata van. Az egyikben a vezérlés a blokkot záró END-en keresztül távozik a blokkból, a másikban a vezérlés egy külső blokkbeli cimkére kerül.

Az ALGOL 60 blokkmüveletei alapján, a taskrendszerek vezérlési szerkezetére vonatkozóan az alábbi ötletekre jutottunk:

7.1. A taskrendszer álljon egyetlen, taskokból felépülő folyamatból. A taskrendszer müködésének és ábrázolásának szabályai a következők:

a. A folyamat pillanatnyi állapotát egy drivereket tartalmazó verem reprezentálja. Rajzban:



b. A verem legalsó drivere a folyamat alapító taskjának drivere. Ha egy a monitornak szóló parancs hatására ez a driver megszünik, akkor megszünik a folyamat is, és ekkor az ALGOL 60 szerkezetü taskrendszer terminál.

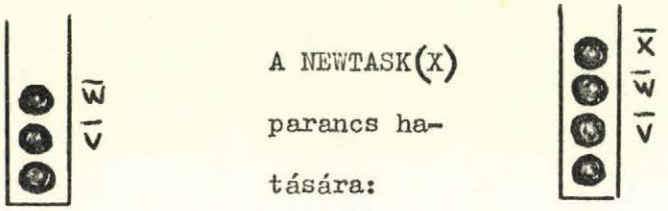
c. A verem tetején lévő driver a folyamat futásképes taskjának drivere. Mivel az ALGOL 60 szerkezetü taskrendszerben egyetlen folyamat van, ezért annak futásképes taskja egyben aktiv is.

7.2. A monitornak szóló vezérlésátadási parancsok a következők:

NEWTASK(X) Az X paraméter egy taskprototipust azonosit. A parancs hatására a monitor egy uj $\overline{X}$ drivert helyez a verem tetejére melyet az X-re vonatkozó információkkal tölti fel.

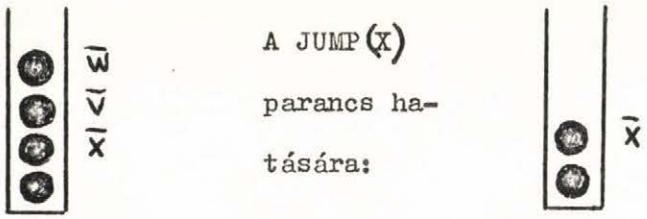
Ezután a TASKCALL eljárás segitségével a taskprototipus alapján létrehoz majd aktivizál a központi memóriában egy uj taskpéldányt.

Példa:



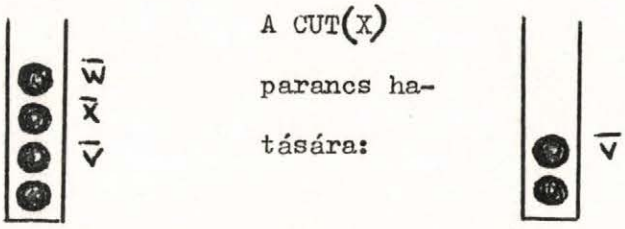
JUMP(X) Az X paraméter egy taskpéldányt azonosit. A parancs hatására a monitor törli a veremből az $\overline{X}$ feletti összes drivert, majd aktivizálja az X taskpéldányt a TASKACTIVATE eljárás segitségével.

Példa:



CUT(X) Az X paraméter egy taskpéldányt azonosit. A parancs hatására a monitor törli a veremből az $\overline{X}$ feletti összes drivert az $\overline{X}$-al együtt, majd a vezérlést annak a tasknak adja a TASKACTIVATE eljárás segitségével, amelynek a drivere a verem tetejére került /ha van ilyen/.

Példa:



8. SIMULA 67 szerkezetü taskrendszerek.

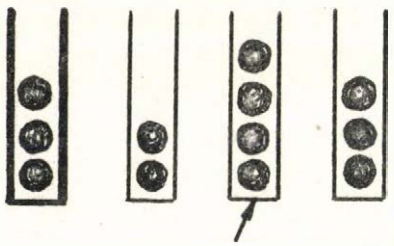
A SIMULA 67, mint az ALGOL 60-at teljes egészében tartalmazó annál magasabb szintü, univerzális programozási nyelv, az ALGOL 60-nál sokkal általánosabb vezérlési szerkezettel rendelkezik: Blokkstrukturái és blokkmüveletei hatékony eszközei a parallel folyamatok szimulációjának illetve realizálásának egyprocesszoros számitógépen.

Mivel a SIMULA 67 egyprocesszoros számitógépet tételez fel, ezért a parallel folyamatok közül mindig pontosan egy használhatja a processzort. Ezt az egy folyamatot aktivnak nevezzük. Ezt kvázi parallel müködési módnak szokás nevezni. A SIMULA 67 tehát valójában nem parallel, hanem csak kvázi parallel folyamatrendszert kezel. A továbbiakban a kvázi parallel folyamatrendszert KPFR-el röviditjük.

A SIMULA 67 folyamatkezelését vizsgálva, a taskrendszerek vezérlési szerkezetére vonatkozóan a következő ötletekre jutottunk.

8.1. A taskrendszer legyen taskokból álló KPFR. Az ALGOL-folyamat egy állapotából eddig a következő driverstrukturát nyertük:

Szemléltessük a taskokból álló KPFR egy állapotát a következő driverstrukturával:



A taskrendszer müködésének és ábrázolásának szabályai:

a. Minden folyamatot egy-egy drivereket tartalmazó verem reprezentál.

b. Mivel egy-egy folyamat akkor születik meg, amikor létrehozzuk alapitó taskját, és akkor szünik meg, amikor töröljük azt, ezért az alapitó taskra való hivatkozással utalhatunk magára a folyamatra is.

c. Egy KPFR pontosan egy kitüntetett, és tetszőleges számu egyenrangu folyamatból áll.

d. A KPFR egyetlen aktiv folyamatát ábráinkon nyil jelöli.

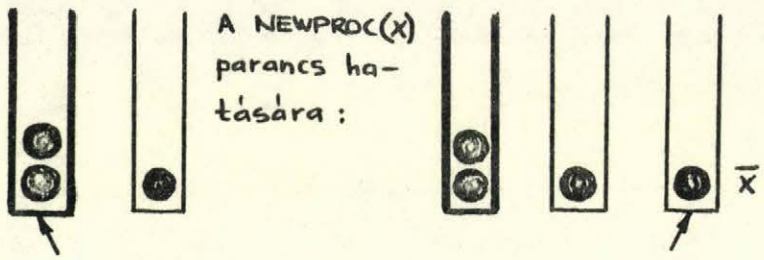
e. A KPFR akkor jön létre, amikor első folyamatát létrehozzuk. Ezt a folyamatot a monitor kitüntetettnek nyilvánitja. A kitüntetett folyamatot reprezentáló vermet vastag vonal jelöli. Minden olyan parancs, melynek eredményeképpen a KPFR elvesztené kitüntetett folyamatát, a KPFR terminálását okozza.

f. A taskrendszer monitora mindig a KPFR aktiv folyamatának futásképes taskját aktivizálja.

8.2. Szükséges az alábbi parancsok bevezetése:

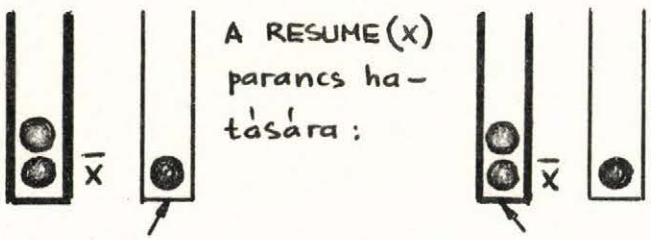
NEWPROC(X) Az X paraméter egy taskprototipust azonosit. A parancs hatására a monitor az uj folyamat reprezentálására egy uj vermet létesit, és elhelyezi benne az $\overline{X}$ drivert. A monitor ezután az uj veremre irányitja a nyilat, majd létrehozza és aktivizálja a központi memóriában az X taskpéldányt a TASKCALL eljárás segitségével a taskprototipus alapján.

Példa:



RESUME(X) Az X paraméter egy folyamatot azonosit, azaz X egy folyamat alapitó taskjának neve. A parancs hatására a monitor a kijelölt folyamatot reprezentáló veremre irányitja a nyilat, majd aktivizálja a kijelölt folyamatot, azaz a folyamat futásképes taskját aktivizálja a TASKACTIVATE eljárás segitségével.

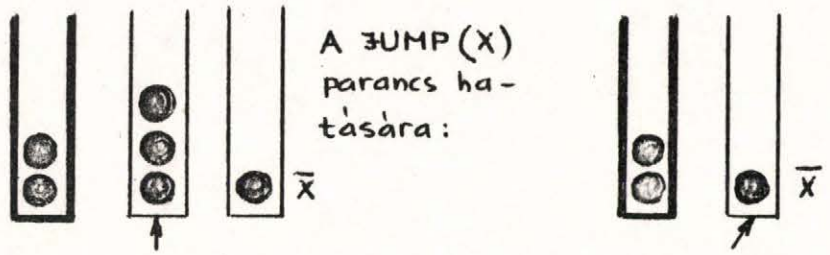
Példa:



Szükséges a két korábban bevezetett parancs általánositása.

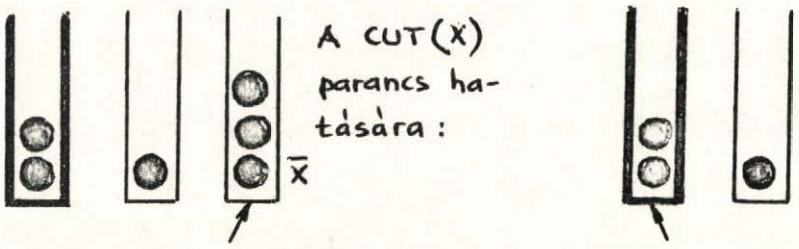
JUMP(X) Ha az X paraméter egy az aktivtól különböző folyamat alapitó taskját azonositja, akkor a parancs hatására a monitor törli a parancsot kiadó aktiv folyamatot, majd végrehajt egy RESUME(X) parancsot.

Példa:

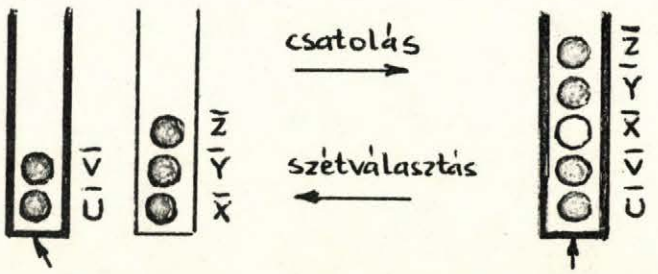


CUT(X) Ha az X paraméter az aktiv folyamat alapitó taskját azonositja, akkor parancs hatására a monitor törli a parancsot kiadó aktiv folyamatot, majd végrehajt a kitüntetett folyamatra egy RESUME parancsot.

Példa:



Szükséges a folyamatok ideiglenes egymáshozcsatolásának, illetve a csatolt folyamatok leválasztásának lehetősége. A csatolást veremtartalmak ideiglenes egymásrahelyezésével, a leválasztást pedig a verem kettévágásával ábrázoljuk:



ATTACH(X) Az X paraméter az aktiv folyamattól különböző, az aktiv folyamathoz csatolandó folyamatot azonositja. A parancs hatására a monitor az aktiv folyamathoz csatolja az X folyamatot, azaz az aktiv folyamat vermébe

helyezi az X vermében lévő drivereket, majd az aktiv folyamat futásképes taskját aktivizálja a TASKACTIVATE eljárás segitségével. A csatolás rendjét a fenti ábra szemlélteti.

DETACH(X) Az X paraméter az aktiv folyamatban egy csatolt folyamat alapitó taskját jelöli. A parancs hatására a monitor a csatolt X folyamatot leválasztja az aktiv folyamatról, majd az aktiv folyamat futásképessé vált taskját aktivizálja a TASKACTIVATE eljárás segitségével. A szétválasztás rendjét a fenti ábra szemlélteti.

8.3. A SIMULA 67 vezérlési szerkezete a fentieknél sokkal bővebb:

a./ A SIMULA általánosabb, többszintes KPFR fogalommal rendelkezik. A többszintesség lényegében azt jelenti, hogy egy KPFR is lehet egy verem eleme, aminek az a jelentése, hogy egy KPFR részrendszere lehet egy másiknak. A többszintes KPFR utasitáskészlete az egyszintes KPFR utasitáskészletének egyszerü általánositása. A többszintes KPFR utasitáskészlete egyszerü számitásokhoz fogalmilag tul sok, magasszintü folyamatkezeléshez illetve szimulációhoz pedig önmagában tul kevés.

b./ A SIMULA azonban szolgáltat egy másfajta, használhatóbb, kényelmesebb, magasszintü folyamatkezelést is, amikor a szimulációs igényeknek megfelelően definiálja a standard szimulációs osztály fogalomkészletét is. A SIMULATION osztály folyamatkezelő utasitásai a KPFR fogalmára és KPFR utasitásokra épülnek, mert a folyamatkezelő utasitások definició szerinti végrehajtása mindig KPFR utasitással /pld. RESUME/ fejeződik be.

A végrehajtás felcserélhetetlen lépései ennek alapján a következők:

1./ Elemzés a folyamatrendszer modelljén.

2./ A megfelelő KPFR utasitás és paraméter kiválasztása, majd végrehajtása.

9. Szimulációs taskrendszerek problémája

Mint mondottuk, a taskrendszerek számára eddig bevezetett KPFR parancskészlet egyszerü taskrendszerek számára tul sok, szimulációs taskrendszerek számára pedig tul kevés. Mivel a SIMULATION osztály a kényelmesebb, magasszintü folyamatkezelést éppen a szimulációk igényeinek megfelelően definiálja, ezért a szimulációs taskrendszerek vezérlési szerkezete és monitora számára a SIMULATION osztály fogalmaiból és elveiből kellene ötleteket meriteni. Nyitott kérdés az, hogy a konkurrens folyamatok szinkronizálásának milyen fogalmaival és elveivel célszerü kiegésziteni azt.

A taskrendszerben a monitornak szóló parancsok kibocsájtásának illetve feldolgozásának sorrendje a következő:

1./ A taskrendszer aktiv taskjában a megfelelő parancs és paraméter kiválasztása, majd kibocsájtása.

2./ A monitorban elemzés a taskrendszer modelljén, majd a parancs végrehajtása.

Mint látjuk, a SIMULA folyamatkezelő utasitásaiban szereplő sorrend éppen ellentétes! Mivel az elemzés és a parancs kiválasztásának sorrendje felcserélhetetlen, ezért a SIMULA visszavezetési eljárása /lásd 8.3./ nem alkalmazható. Éppen ezért okoz különösen nehéz problémát az elvárásoknak megfelelő /a reális igényeket lefedő, egyszerü, kifejező, biztonságos, magasszintü, stb./ parancskészlet, és monitorbeli modell kialakitása.

Irodalomjegyzék

1 O-J. Dahl, B. Myhrhaug, K. Nygaard:
SIMULA 67 Common Base Language.
Norwegian Computing Center, OSLO, 1970.

2 O-J. Dahl, B. Myhrhaug:
SIMULA 67 Implementation Quide.
Norwegian Computing Center, OSLO, 1973.

3 Gáspár A., Visontay Gy., Csáki P.:
A CDC 3300 SIMULA/MASTER uj eszközei felhasználói taskrendszerek számára
CDC 3300 Felhasználói Ismertetők 11.,
MTA SZTAKI, Budapest, 1977.

A szerzők köszönetet mondanak Sztanó Tamásnak értékes megjegyzéseiért.

Quasi parallel task systems

A. Gáspár, Gy. Visontay, P. Csáki

Computer and Automation Institute Hungarian Academy of Sciences

The paper presents a general system which makes it possible to develop task systems having quasi parallel operating structures.

Псевдо - параллелные системы

А. Гашпар, Г. Вишонтаи, П. Чаки

В стате писывается общая система осуществляешая создание системы задач с псевдо - параллелным режимом.

Zur Analyse des Echtzeitbetriebs von Prozeßrechnersystemen

Gerhard Bergholz

Technische Universität Dresden

1. Echtzeitbetrieb

Prozeßrechnersysteme arbeiten im Echtzeitbetrieb. Dabei verstehen wir unter Echtzeitbetrieb den schritthaltenden Ablauf der Prozesse im Prozeßrechnersystem mit den zu überwachenden oder zu steuernden Streckenprozessen. Wir führen die Echtzeitbedingung

$$T_r \leq T_{rz} \qquad (1)$$

ein, die engibt ob der Echtzeitbetrieb gesichert ist. Dabei wird mit T_r die Reaktionszeit eines Echtzeitauftrags und mit T_{rz} die zulässige Reaktionszeit bezeichnet. Die Zulässige Reaktionszeit T_{rz} ist durch die Streckenprozesse bestimmt.

Wir betrachten einen Strom von Echtzeitauftägen, die eine Reaktionszeit T_r besitzen. Dabei nehmen wir an, daß die Reaktionszeit T_r eine Zufallsgröße ist. Für diesen Strom führen wir die Echtzeitwahrscheinlichkeit ein. Die Echtzeitwahrscheinlichkeit P_e gibt an mit welcher Wahrscheinlichkeit die Ungleichung (1) durch die Echtzeitaufträge des betrachteten Stromes eingehalten wird. Es gilt also

$$P_e = P\left\{T_r \leq T_{rz}\right\}. \qquad (2)$$

Die Verteilungsfunktion der Reaktionszeit T_r ist mit

$$R(t) = P\left\{T_r = t\right\} \tag{3}$$

definiert. Aus den Ansätzen (2) und (3) folgt

$$P_e = R(T_{rz}) \ . \tag{4}$$

Somit kann die Echtzeitwahrscheinlichkeit P_e durch Einsetzen von T_{rz} in die Verteilungsfunktion der Reaktionszeit bestimmt werden.

Die Bestimmung der Verteilungsfunktion der Reaktionszeit eines Stroms von Echtzeitaufträgen ist folglich eine wichtige Aufgabe für die Analyse des Echtzeitbetriebs eines Prozeßrechnersystems.

2. Zur Bestimmung der Reaktionszeitverteilung

Wir betrachten die Laplace-Stieltjes-Transformierte $R^*(s)$ der Reaktionszeitverteilung $R(t)$ mit dem Ansatz

$$R^*(s) = \int_0^\infty e^{-st}.dR(t). \tag{5}$$

Zur Unterscheidung bezeichnen wir $R(t)$ als Verteilungsfunktion im Zeitbereich und R^* als Verteilungsfunktion im Bildbereich.

Für die Reaktionszeit eines Stromes von Echtzeitaufträgen gilt allgemein

$$T_r = T_v + T_s \ . \tag{6}$$

Hierbei ist T_v die Auftragsverweilzeit und T_s die Auftragsverschiebungszeit (s./1/). Wenn mit $V^*(s)$ die Verteilungsfunktion der Verweilzeit und mit $S^*(s)$ die Verteilungsfunktion der Verschiebungszeit im Bildbereich angenommen werden, dann gilt für die Verteilungsfunktion der Reaktionszeit im Bildbereich

$$R^*(s) = V^*(s).S^*(s). \tag{7}$$

In manchen Fällen ist

$$T_s = 0, \tag{8}$$

woraus folgt

$$S^*(s) = 1. \tag{9}$$

Für die von uns betrachteten Fälle mit $T_s=0$ gilt der Ansatz

$$S^*(s) = \frac{1-e^{-sT_a}}{s}. \tag{10}$$

Dieser Ansatz wird in den weiteren Untersuchungen genutzt.

Außerdem muß noch als Grundlage für die Bestimmung der Reaktionszeitverteilung die Verweilzeitverteilung ermittelt werden.

3. Verweilzeitelemente und ihre Kopplung

Wir nehmen jetzt an, daß in die Bedienung eines Stromes von Echtzeitaufträgen durch das Prozeßrechnersystemen Verweilzeitelemente eingehen. Dabei habe das Verweilzeitelement Nummer i die Verweilzeitverteilung: im Zeitbereich $V_i(t)$,(i=1, 2,....n) und im Bildbereich $V^*(s)$, (i=1,2,...,n). Die Tafel 1 enthält die symbolische Darstellung der Verweilzeit- und Kopplungselemente, mit deren Hilfe ein Forderungslaufplan (s./2/) für die Bestimmung der Verteilungsfunktion der Auftragsverweilzeit aufgestollt werden kann.

Für die Erzeugung von Forderungen existieren eine äußere und eine innere Quelle. In einer äußeren Quelle werden wiederholt Forderungen (Echtzeitaufträge) mit einem Ankunftsabstand T_a erzeugt. Beim Durchlauf einer Vorderung durch eine innere Quelle wird eine zweite Forderung erzeugt (Forderungsspaltung, s. auch /3/). Beide Forderungen laufen anschließend parallel weiter.

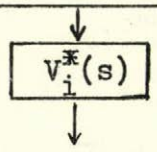
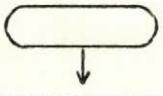
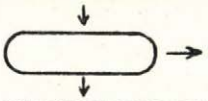
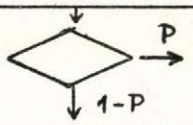
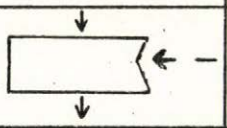
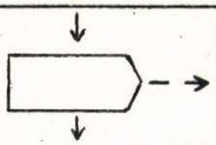
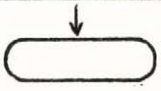
Eine Forderung, die bei einem Verzweigungselement eintrifft, wird mit der Wahrscheinlichkeit 1-P geradeaus weitergeleitet und mit der Verzweigungswahrscheinlichkeit P abgezweigt.

Ein Suspendierungselement dient als Tor für die ankommenden Forderungen. In diesem Element wird eine ankommende Forderung solange suspendiert (aufgehalten), bis aus einem Fortsetzungselement das entsprechende Fortsetzungssignal eintrifft.

Mit dem Eintreffen einer Forderung in einer Senke wird diese Forderung vernichtet.

Wir betrachten jetzt das Beispiel eines Forderungslaufplanes, der aus Verweilzeitelementen aufgebaut ist. Dabei gehen wir von der Wechselwirkung eines Applikationsprogrammes mit einem parallelen Informationsaustauschkanal aus (s. auch /4). Für diesen Fall gilt der in Bild 1 dargestellte Forderungslaufplan. Dabei ist der Abschnitt links von der Strichpunktlinie dem Applikationsprogramm zugeordnet und der Abschnitt rechts von der Strichpunktlinie dem Informationsaustauschkanal. Die Verweilzeitelemente mit den Verweilzeitverteilungen $V_1^*(s)$, $V_2^*(s)$, $V_3^*(s)$, $V_4^*(s)$ entsprechen also Programmlaufzeiten und Programmwartezeiten, während das Verweilzeitelement $V_5^*(s)$ der Übertragungszeit eines Übertragungswortes entspricht.

Tafel 1: Elemente für den Aufbau eines Verweilzeit-Forderungslaufplanes

Symbol	Elementtyp
	Verweilzeitelement Nummer i (i=1, 2, ...,n)
	Äußere Quelle
	Innere Quelle
	Verzweigungselement
	Suspendierungselement
	Fortsetzungselement
	Senke

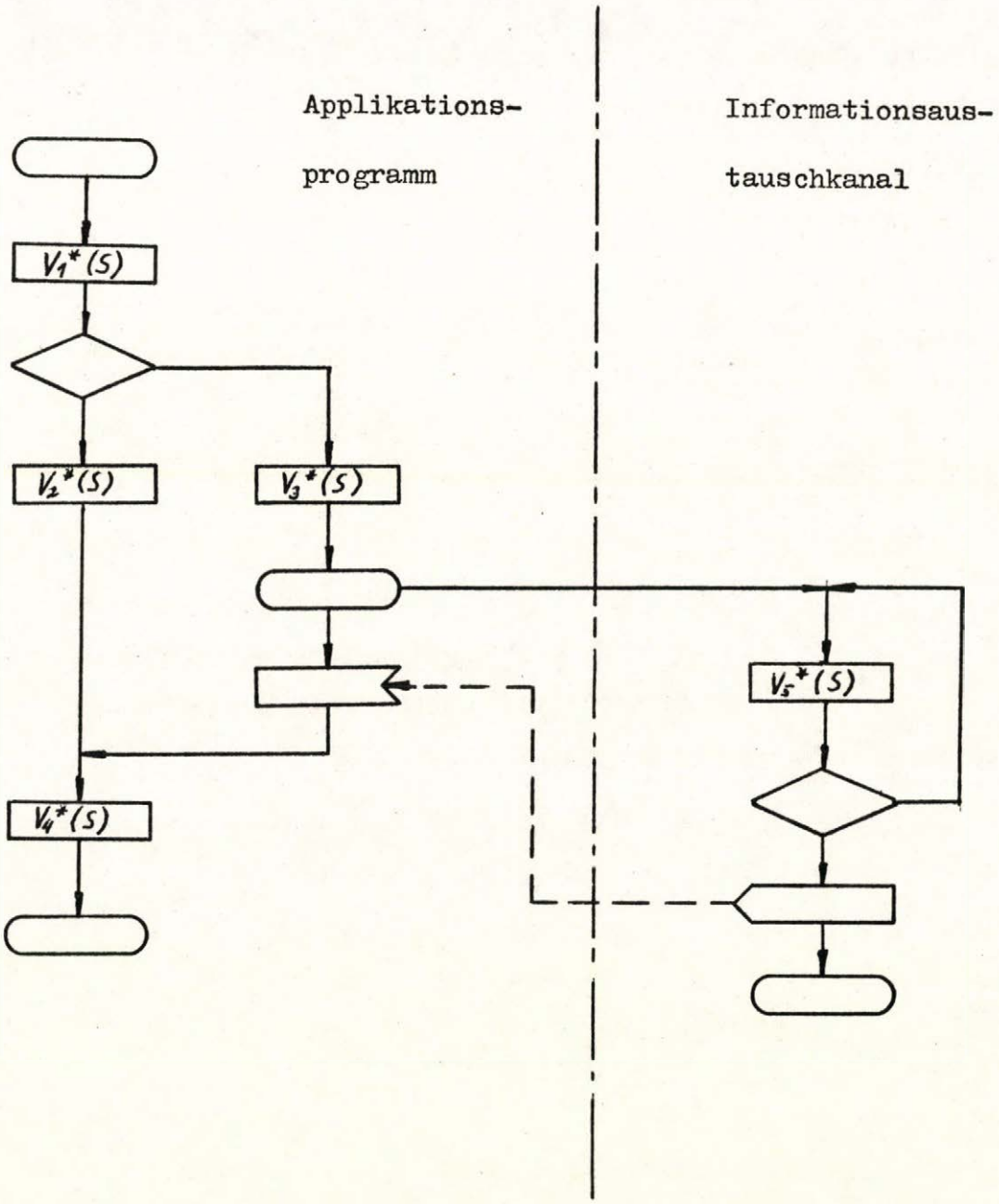


Bild 1: Verweilzeit-Forderungslaufplan für die Wechselwirkung eines Applikationsprogrammes mit einem parallelen Informationsaustauschkanals

4. Verteilungsfunktionen elementerer Kopplungsformen

Wir betrachten jetzt folgende elementaren Kopplungsformen

- die Serienkopplung
- die Verzweigungskopplung
- die Schleifenkopplung
- die Parallelkopplung und
- die Synchronisation.

Mit Hilfe dieser Kopplungsformen lassen sich eine Vielzahl möglicher Verweilzeit- Forderungslaufpläne aufbauen.

Für die oben genannten elementaren Kopplungsformen sind in Tafel 2 die Formeln zur Bestimmung der Verweilzeitverteilung des gekoppelten Abschnittes aus den Verweilzeitverteilungen der Verweilzeitelemente angegeben. Für die Schleifenkopplung wurde mit P_i die Wahrscheinlichkeit, daß die Schleife i-mal durchlaufen wird, angenommen. Daraus ergibt sich für den Fall, daß die Schleife genau l-mal durchlaufen wird

$$P_i = \begin{cases} 0 \text{ für } i=1,2,\ldots,l-1 \\ \\ 1 \text{ für } i=l. \end{cases} \qquad (11)$$

Wenn wir dagegen für jeden Durchlauf der Schleife an der Verzweigungsstelle eine konstante Verzweigungs-

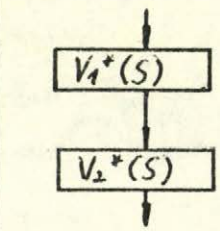
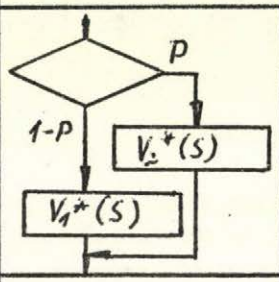
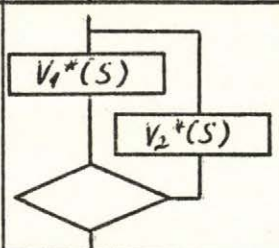
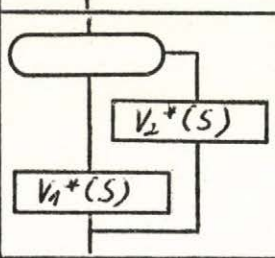
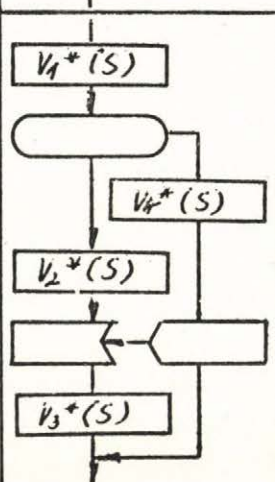
wahrscheinlichkeit P ansetzen, dann erhalten wir

$$P_i = (1-P).P^{i-1}. \qquad (12)$$

Bei der Parallelkopplung wurde angenommen, daß die beiden Zufallsgrößen T_{v1} und T_{v2} statistisch unabhängig voneinander sind. Da die Formel für die Parallelkopplung für die Verteilungsfunktionen im Zeitbereich gilt, muß bei gemeinsamer Nutzung dieser Formel mit solchen für die Verteilungsfunktion im Bildbereich die Laplace-Stieltjes-Transformation und die entsprechende Rücktransformation ausgeführt werden. Die bisher praktisch ausgeführten Untersuchungen zeigen, daß die dabei auftretenden Probleme geringer sind, als das zunächst aussieht.

Der Synchronisation ist im angeführten Fall eine Parallelkopplung untergeordnet. Das hat zur Folge, daß hier die Laplace-Stieltjes-Transformation ausgeführt werden muß.

Tafel 2: Zur Bestimmung der Verweilzeitverteilung elementarer Kopplungsformen

Serien-kopplung		$V^*(s) = V_1^*(s).V_2^*(s)$ (13)
Verzweigungs-kopplung		$V^*(s) = P.V_1^*(s) + (1-P).V_2^*(s)$ (14)
Schleifen-kopplung		$V^*(s) = \sum_{i=1} P_i [V_1^*(s)]^i [V_2^*(s)]^{i-1}$ (15)
Parallel-kopplung		$V(t) = V_1(t).V_2(t)$ (16) (im Zeitbereich)
Synchroni-sation		$V^*(s) = V_1^*(s).V_{24}^*(s).V_3^*(s)$ (17) $V_{24}^*(s) = e^{-st}.dV_{24}(t)$ (18) $V_{24}(t) = V_2(t).V_4(t)$ (19)

5. Beispiel für die Bestimmung der Reaktionszeitverteilung

In der Arbeit (5) wurde für den Anwendungsfall einer Meßwerterfassung und -verarbeitung der Erwartungswert der Reaktionszeit analysiert. Jetzt soll für dieses Beispiel die Verteilungsfunktion der Reaktionszeit untersucht werden. Durch Umformung kann der Verweilzeit-Forderungslaufplan, der in Bild 2 dargestellt ist, für diese Beispiel gewonnen werden.

Durch schrittweise Anwendung der in der Tafel 2 angegebenen Formeln, erhalten wir die Verweilzeitverteilung der Echtzeitaufträge, die den in Bild 2 angegebenen Forderungslaufplan durchlaufen.

Unter Nutzung des dabei gewonnenen Ergebnisses für die Verweilzeitverteilung und der Formel (1o) für die Verschiebungszeitverteilung erhalten wir

$$R(s) = \frac{1-e^{-sT_a}}{s} \cdot \sum_{j=1}^{5} P_j . \exp[-s(n.T_{bn}+T_{bm})]. \qquad (2o)$$

Dabei gilt

$$T_{bn} = T_{b3} + T_{b4}, \qquad (21)$$

$$T_{bm} = T_{b1} + T_{b2} + T_{b3} + T_{b5} + T_{b6}, \qquad (22)$$

$$n = \frac{E(M)}{\sum_{j=1}^{5} P_j . j} \cdot \qquad (23)$$

Die auf diese Weise gewonnene Verteilungsfunktion ist für konkrete Parameterwerte in Bild 3 dargestellt. In dieses Bild ist auch die untere Grenze für die zulässige Reaktionszeit $T_{rz\ P_e=1}$ eingezeichnet.

Mit Hilfe der Formel (2o) kann auch der Erwartungswert der Reaktionszeit $E(T_r)$ ermittelt werden. Dazu wird die negative Ableitung der Reaktionszeitverteilung gleich 0 gesetzt.

In Bild 4 wurden Ergebnisse für den Erwartungswert $E(T_r)$ in Abhängigkeit vom Ankunftsabstand T_a des Ankunftstromes aufgetragen. Dabei wurde für den Informationsaustauschkanal ein M/M/1-Bedienungsmodell, ein D/M/1-Bedienungsmodell und ein G/G/ -Modell betrachtet und vergliechen. Außerdem sind in Bild 4 Simulationsergebnisse zum Vergleich eingetragen. Dies Simulationsergebnisse wurden mit Hilfe des VOPS SIMDIS, das eine blockorientierte Sprache einschließt, erhalten.

Für große Ankunftsabstände, denen kleinere Auslastungsgrade des Informationsaustauschkanals entsprechen, stimmen alle mit verschiedenen Bedienungsmodellen gewonnenen Ergebnisse gut überein und entsprechen auch den Simulationsergebnissen.

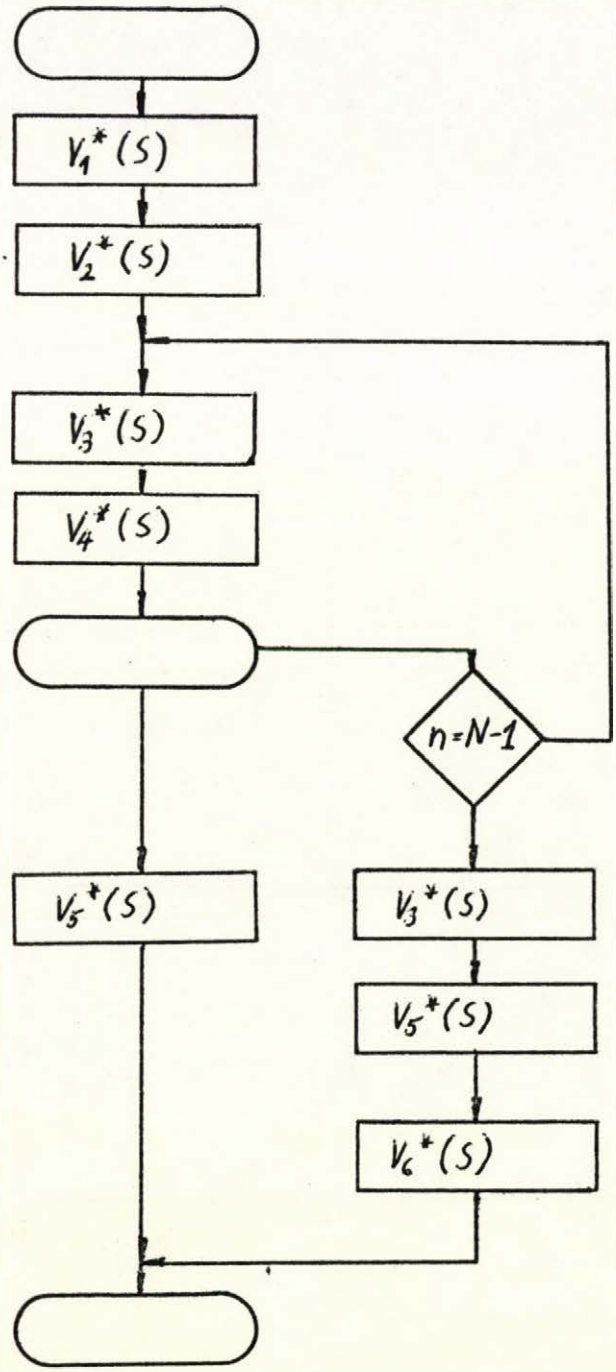


Bild 2: Verweilzeit-Forderungslaufplan eines Beispiels

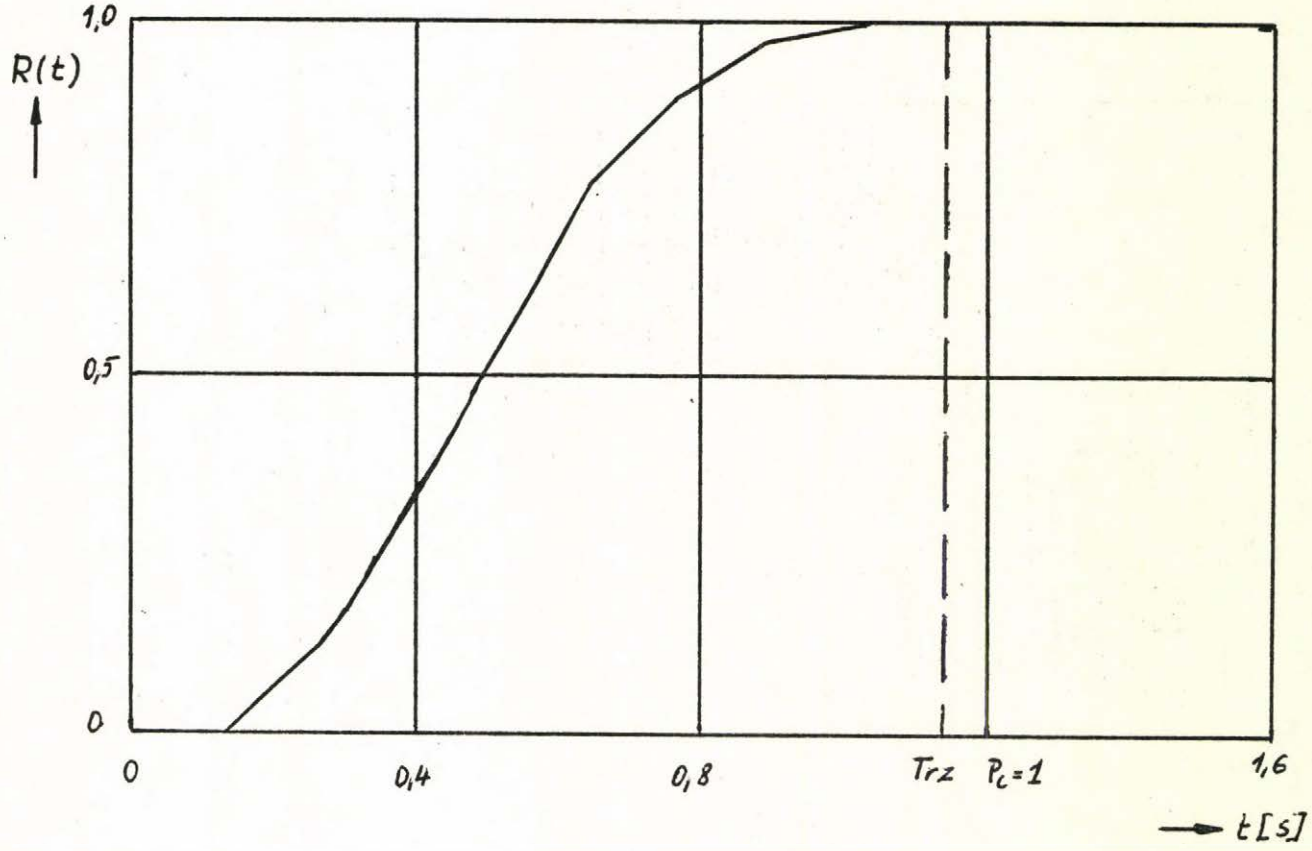


Bild 3: Verteilungsfunktion der Reaktionszeit für ein Beispiel

Für kleine Ankunftsabstände, bei denen der Informationsaustauschkanal fast vollständig ausgelastet ist, gibt es große Unterschiede zwischen den verschiedenen Modellansätzen. Leider wächst in diesem Bereich der statistische Fehler der Simulationsuntersuchung auch stark an, sodaß mit den zur Zeit vorliegenden Simulationsergenbissen keine Entscheidung darüber möglich, welcher theoretische Ansatz genauere Ergebnisse liefert.

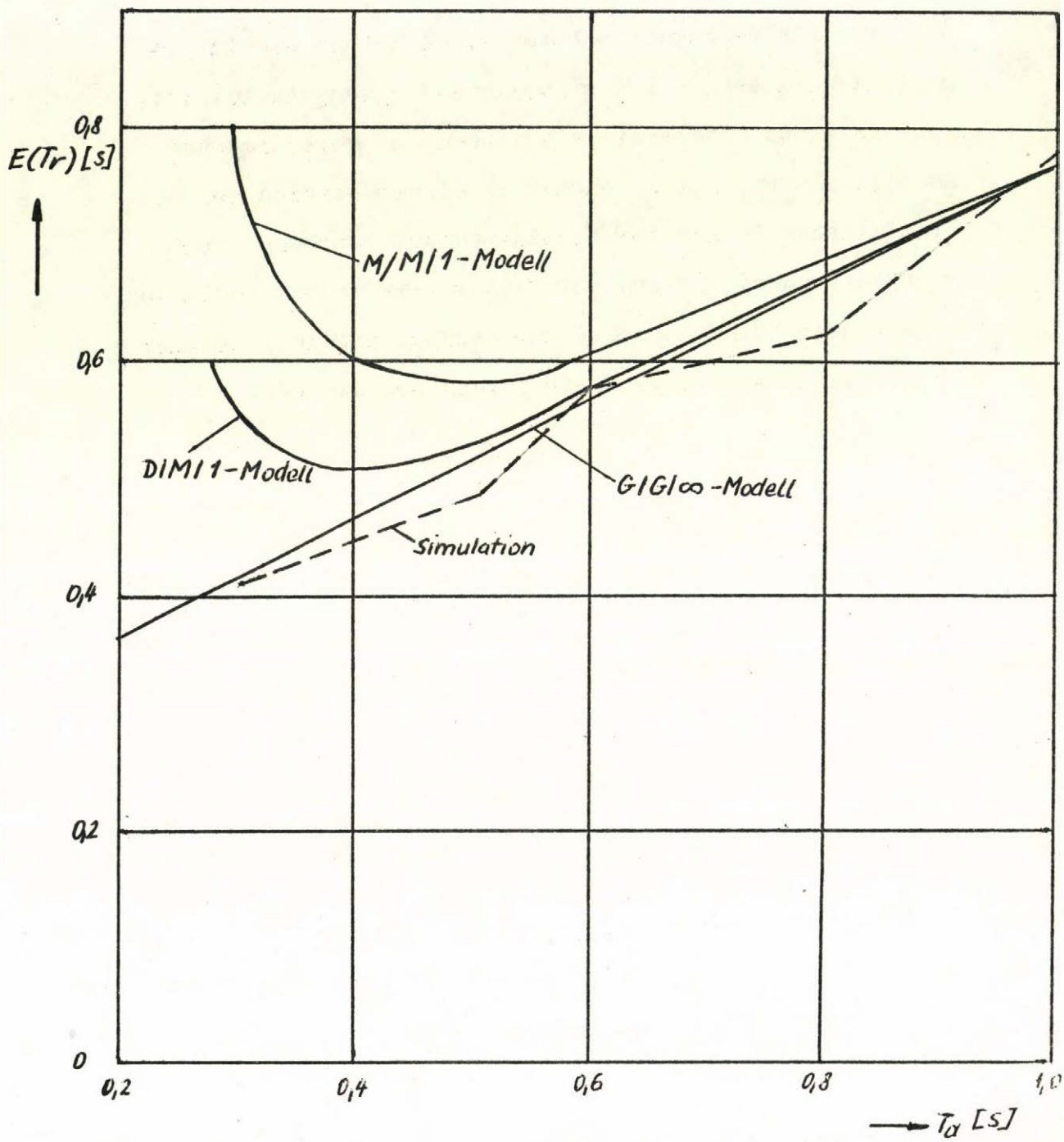


Bild 4. Einfluß des Ankunftsabstands der Echtzeitaufträge auf die Reaktionszeit

Literaturverzeichnis:

1. Bergholz, G.: Zur Bestimmung der Reaktionszeit von Prozeßrechenanlagen. mar 19 (1976) H. 4

2. Bergholz, G.: Zur Ermittlung der Forderungsstrominten-sitäten in einem Echtzeitoperations-system für Prozeßrechenanlagen. Wiss. Zeitschrift der Technischen Uni-versität Dresden, 24 (1975) H. 3/4

3. Bergholz, G.: Zur Analyse der Multiprogrammbearbeitung in einer Prozeßrechenanlage. Rechen-technik/Datenverarbeitung 12(1975), Beiheft 3

4. Bergholz, G.: Echtzeitmodell für den Informations-austausch zwischen der Zentraleinheit und der Peripherie eines Prozeßrechnern. msr 18 (1975) H. 11

Real-time folyamatok analizise

Bergholz G.

A dolgozat real-time rendszerek valószinüségszámitási modelljével foglalkozik, mely különböző, a dolgozatban megadott absztrakt elemekből épithető fel.

Analysis of real-time processes

G. Bergholz

The paper contains a probabilistic model of real-time systems consisting of abstract elements defined in the paper.

Р Е З Ю М Е

Анализ процессов, работающих в реальном масштабе времени

Г. Бергольц

В работе рассматривается случайная система процессов.