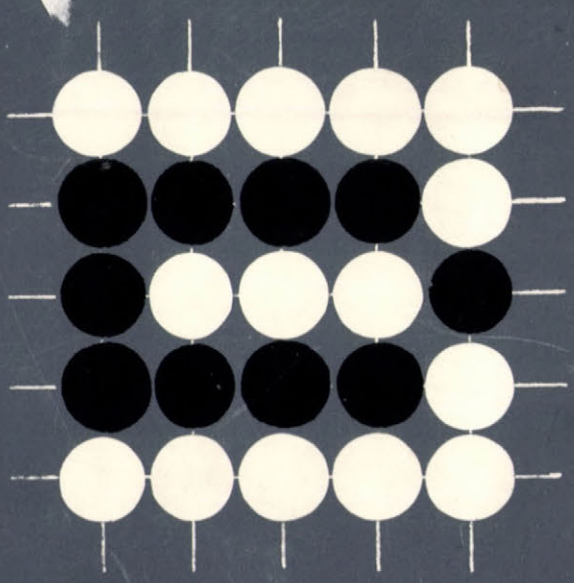
közlemények 18/1977

MTA Számítástechnikai és Automatizálási Kutató Intézet Budapest



MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZET

KÖZLEMÉNYEK

ISBN 963 311 038 6

1977. JANUÁR

Felelős kiadó:
Dr. Vámos Tibor
igazgató

OPERÁCIÓS RENDSZEREK ELMÉLETE

Téli Iskola, Visegrád

MTA Számítástechnikai és Automatizálási Kutató Intézet
MTA Számítástudományi Bizottsága

Technikai szerkesztő:

Solt Jánosné

MTA Számítástechnikai és Automatizálási Kutató Intézete

TARTALOMJEGYZÉK

CONTENTS

ОГЛАВЛЕНИЕ

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

DETERMINISTIC SEQUENCING: COMPLEXITY AND OPTIMAL ALGORITHMS

E. G. Coffman, Jr.*
The Pennsylvania State University

I. Introduction

In the past several years interest and new results in the theory of deterministic scheduling have mounted at an increasing rate. This paper is an attempt to represent the current position of the field in terms of research into schedule length and mean flow time minimization problems. We describe theoretical results for sequencing problems arising mainly in computer and job-shop environments. However, the models are simple in structure and are consequently meaningful in a very large variety of applications.

Briefly, the general model studied assumes a set of tasks or jobs and a set of resources to be used in their execution or servicing. In all cases the models are deterministic in the sense that the information describing tasks is assumed known in advance. This information includes task execution times, operational precedence constraints, deferral costs, and resource requirements. The sequencing problems examined include not only the minimization of schedule-lengths and mean time-in-system (weighted by deferral costs), but also a number of closely related problems such as scheduling to meet due-dates or deadlines. The results presented include efficient optimal algorithms and mathematical descriptions of the complexity of sequencing problems.

---

*This research was partially supported by Institut de Recherche d'Informatique et d'Automatique, France, and by NSF grant GJ-28290.

Computers arise in the subject matter in at least three ways. Firstly, they represent an almost universal job-shop for our purposes. The appearance of virtually all of the problems we analyze can be observed or envisioned in the design or operation of general-purpose computer systems, although the prime importance of specific problems may exist in other applications. Secondly, computers must be considered in the implementation of the enumerative and iterative approaches to sequencing problems. Finally, the field of Computer Science is the origin of the complexity theory which we apply to problems of sequence.

We emphasize that our interest is almost wholly mathematical, with very little recourse to discussions of pragmatics. The applicability (and, of course, inapplicability) of the results will be quite evident in virtually all cases, owing primarily to the simplicity of the models. The reader is referred to [CMM] for insights into the general problems in practice and the many features of such problems that extend the models examined here but for which comparable results are not known (see also [Ba]).

This paper presents virtually all the theoretical results in [Ba] and [CMM] that concern our problems of deterministic scheduling theory. They form the background for the new results. Additional background material concerned with computer sequencing problems can be found in a more recent text [CD] on operating-systems theory. For recent survey papers dealing with many of the subjects of this paper, the reader is referred to [G3], [C1], [B1], and [BLR].

II. A General Model

The scheduling model, from which subsequent problems are drawn, is

described by considering in sequence the resources, task systems, sequencing constraints, and performance measures.

Resources. In the majority of the models studied, the resources consist simply of a set $P = \{P_1,\ldots,P_m\}$ of processors. Depending on the specific problem, they are either identical, identical in functional capability but different in speed, or different in both function and speed.

In the most general model there is also a set of additional resource types $R = \{R_1,\ldots,R_s\}$, some (possibly empty) subset of which is required during the entire execution of a task on some processor. The total amount of resource of type R_j is given by the positive integer m_j. In the computer application, for example, such resources may represent primary or secondary storage, input/output devices, or subroutine libraries. Although it is possible to include the processors in R, it is more convenient to treat them separately because

1) they will constitute a resource type necessarily in common with all tasks (although two different tasks need not require the same processors), and

2) they are discretized with the restriction that a task can execute on at most one processor at a time.

Task Systems. A general task system for a given set of resources can be defined as the system $(T,\prec,[\tau_{ij}],\{R_j\},\{w_j\})$ as follows:

1. $T = \{T_1,\ldots,T_n\}$ is a set of tasks to be executed.
2. $\prec$ is an (irreflexive) partial order defined on T which specifies operational precedence constraints. That is, $T_i \prec T_j$ signifies that T_i must be completed before T_j can begin.
3. $[\tau_{ij}]$ is an $m \times n$ matrix of execution times, where $\tau_{ij} > 0$ is the

time required to execute T_j, $1 \le j \le n$, on processor P_i, $1 \le i \le m$. We suppose that $\tau_{ij} = \infty$ signifies that T_j cannot be executed on P_i and that for each j there exists at least one i such that $\tau_{ij} < \infty$. When all processors are identical we let τ_j denote the execution time of T_j common to each processor.

4. $R_j = [R_1(T_j), \ldots, R_s(T_j)]$, $1 \le j \le n$, specifies in the ith component, the amount of resource type R_i required throughout the execution of T_j. We always assume $R_i(T_j) \le m_i$ for all i and j.

5. The weights w_i, $1 \le i \le n$, are interpreted as deferral costs (or more exactly cost rates), which in general may be arbitrary functions of schedule properties influencing T_i. However, the w_i are taken as constants in the models we consider. Thus the "cost" of finishing T_i at time t is simply $w_i t$.

This formulation contains far more generality than we intend to embrace, but each problem studied can be represented as a special case of the model. One particular restriction worth noting is the limitation on operational precedence. We cannot, for example, represent loops in computer programs modeled as task systems. Note that the partial order $\prec$ is conveniently represented as a directed, acyclic graph (or dag) with no (redundant) transitive arcs. Unless stated otherwise, we assume $\prec$ is given as a list of arcs in such a graph. In general, however, the way in which a partial order is specified in a given problem may influence the complexity of its solution. (We return to this point later.)

In Figure 1, an example system is shown, where the notation T_i/τ_i is introduced for labeling vertices. As one might expect, heavy use is made of graphical methods for defining task systems, rather than defining them as appropriate five-tuples.

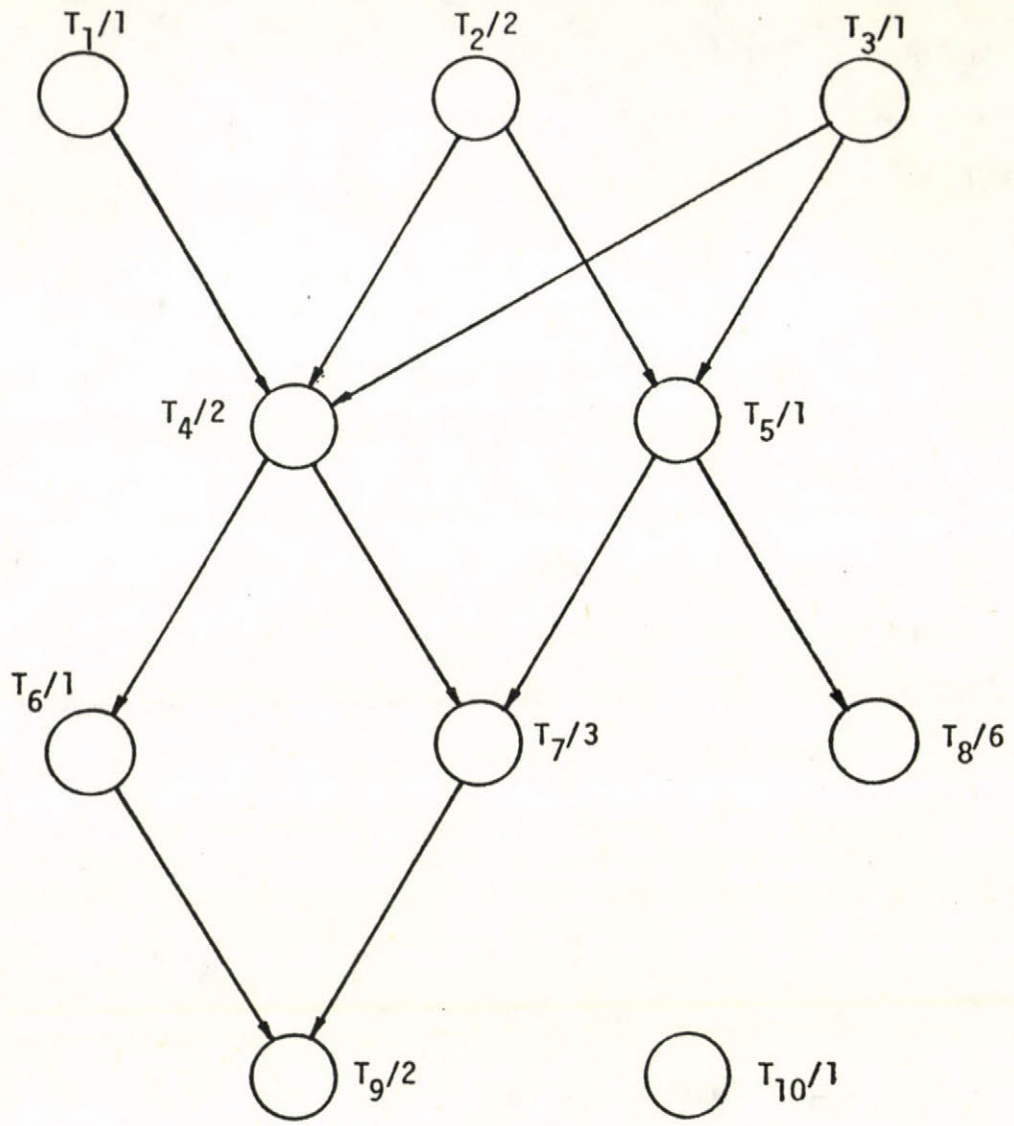


Figure 1. A dag representation of $(T, \prec, \{\tau_i\})$.

Notation and properties:

1. Acyclic.
2. No transitive edges: (T_1, T_6) would be such an edge.
3. T_1, T_2, T_3, T_{10} are initial vertices; T_8, T_9, T_{10} are terminal vertices.
4. For example, T_7 is a successor of T_1, T_2, T_3, T_4, T_5 but an immediate successor of only T_4, T_5; T_5 is a predecessor of T_7, T_8, T_9 but an immediate predecessor of only T_7, T_8.
5. Levels:

8	9	8	7	7	3	5	6	2	1
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}

6. Critical paths: T_2, T_5, T_8 and T_2, T_4, T_7, T_9.

In the following we use a number of more or less common terms concerning dags. In particular, a path of length k from T to T' in a given graph G is a sequence of vertices (tasks) $T_{i_1},\ldots,T_{i_k}$ such that $T = T_{i_1}, T' = T_{i_k}$ $(k \geq 1)$ and $(T_{i_j}, T_{i_{j+1}})$ is an arc in G for all $1 \leq j \leq k-1$. Moreover, if such a path exists, T will be called a predecessor of T' and T' a successor of T. If $k = 2$ the terms immediate predecessor and immediate successor will be used. Initial vertices are those with no predecessors, and terminal vertices are those with no successors. The graph forms a forest if either each vertex has at most one predecessor, or each vertex has at most one successor. If a forest has in the first case exactly one vertex with no predecessors, or in the second case, exactly one vertex with no successors, it is also called a tree. In either case, the terms root and leaf have the usual meaning. The level of a vertex T is the sum of the execution times associated with the vertices in a path from T to a terminal vertex such that this sum is maximal. Such a path is called a critical path if the vertex T is at the highest level in the graph.

Sequencing Constraints. By "constraint" we mean here a restriction of scheduling algorithms to specific (though broad) classes. Two main restrictions are considered.

1. Nonpreemptive scheduling: with this restriction a task cannot be interrupted once it has begun execution; that is, it must be allowed to run to completion. In general, preemptive scheduling permits a task to be interrupted and removed from the processor under the assumption that it will eventually receive all its required execution time, and there is no loss of execution time due to preemptions (i.e., preempted tasks resume execution from the point at which they were last preempted).

2. List scheduling [G1]: in this type of scheduling an ordered list of the tasks in T is assumed or constructed beforehand. This list is often called the priority list. The sequence by which tasks are assigned to processors is then decided by a repeated scan of the list. Specifically, when a processor becomes free for assignment, the list is scanned until the first unexecuted task T is found which is ready to be executed; that is, the task can be executed on the given processor, all predecessors of T have been completed, and sufficient resources exist to satisfy $R_i(T)$ for each $1 \le i \le s$. This task is then assigned to execute on the available processor. We assume the scan takes place instantaneously, and if more than one processor is ready for assignment at the same time, they are assigned available tasks in the order P_1 before P_2 before P_3, etc.

Before discussing performance measures, let us illustrate the means by which schedules are usually represented graphically, assuming $s = 0$. We use the type of timing diagram illustrated in Fig. 2 for the task system shown in Fig. 1. In the obvious way the number of processors determines the number of horizontal lines which denote time axes. The hatching shown in the figure represents periods during which processors are idle. The symbols $s_i(S)$ and $f_i(S)$ will denote, respectively, the start and finishing times of T_i.

For problems assuming additional resources, we will have occasion to draw timing diagrams only for $s = 1$. In this case the vertical axis denotes the amount of additional resource required, the number of vertical segments being bounded by the given number of processors. An example is given shortly (Fig. 4b).

The timing diagrams of Fig. 2 give an informal and intuitive notion of schedule. Somewhat more formally, a schedule can be defined as a suitable mapping that in general assigns a sequence of one or more disjoint execution intervals in $[0,\infty)$ to each task such that

1. Exactly one processor is assigned to each interval.
2. The sum of the intervals is precisely the execution time of the task, taking into account, if necessary, different processing rates on different processors.
3. No two execution intervals of different tasks assigned to the same processor overlap.
4. Precedence and additional-resource usage constraints are observed.
5. There is no interval in $[0,\max\{f_i\}]$ during which no processor is assigned to some task (i.e., a schedule is never allowed to have all processors idle when uncompleted tasks exist).

For nonpreemptive schedules there is exactly one execution interval for each task, and for list schedules we further require that no processor can be idle if there is a task ready and able to execute on it. We do not attempt to further formalize the notion of schedules--a wholly mathematically definition is unnecessary and very elaborate for the general model.

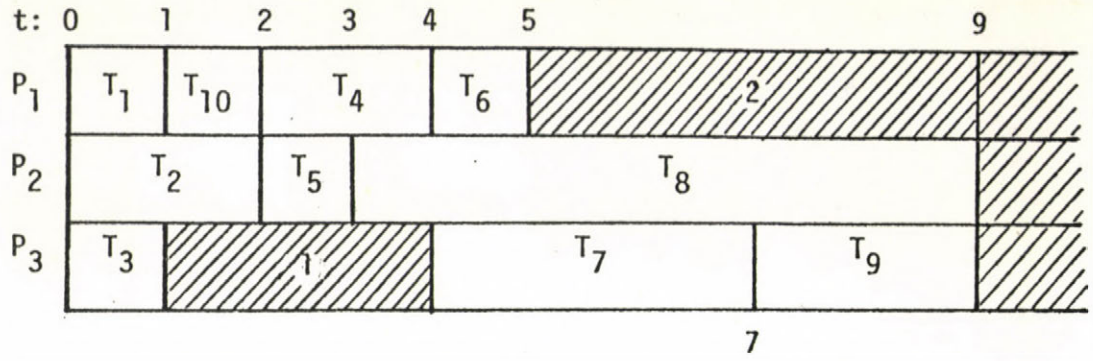


Figure 2. Example timing diagram for Fig. 1. Schedule with $m = 3$.

Performance Measures. Two principal measures of schedule performance that we consider are the schedule-length or maximum finishing (or flow) time

$$\omega(S) = \max_{1 \le i \le n} \{f_i(S)\} \tag{1}$$

and the mean weighted finishing (or flow) time

$$\bar{\omega}(S) = \frac{1}{n} \sum_{i=1}^{n} w_i f_i(S) \ . \tag{2}$$

The basic problems, therefore, are to find efficient algorithms for the minimization of these quantities over all schedules S, drawn perhaps from a specific class of schedules as defined earlier.

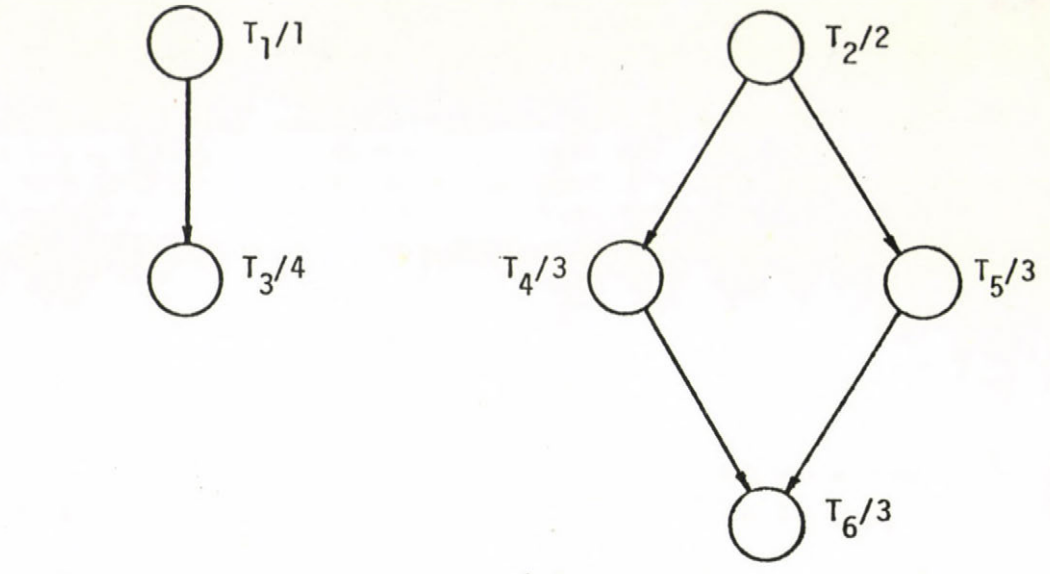
At this point it is convenient to illustrate that preemptive, nonpreemptive, and list scheduling disciplines can be distinct in terms of minimum schedule length and mean weighted flow time and that the "power" of these disciplines decreases in the order given. A graph of a task system, a minimum-length preemptive (i.e., unrestricted) schedule, a minimum-length nonpreemptive schedule, and a minimum-length list schedule appear in

Fig. 3a, b, c, and d, respectively. (Verifying the optimality of the schedules is not difficult). Note that for $w_i = 1$ $(1 \le i \le n)$ the mean weighted flow-time performance is optimal in each case and concurs with the ordering in terms of minimum schedule lengths just given.

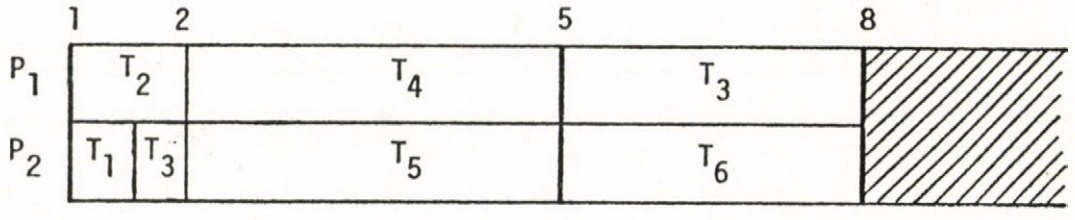
Even for $\prec$ empty, preemptive schedules can be shorter than non-preemptive schedules (e.g., consider $m = 2$ and three unit length tasks). However, for nonpreemptive scheduling on identical processors, a minimum-length list schedule is a minimum-length nonpreemptive schedule when $\prec$ is empty. For mean weighted flow-time (non-negative weights) on identical processors, the optimal list schedule is an optimal preemptive schedule when $\prec$ is empty, thus removing any distinction between the disciplines in these circumstances.

A good many of the results, especially those related to problem complexity, can be readily extended to a number of other performance measures of interest in job-shop or computer sequencing. For example, suppose we have identical processors and we define $W_i(S) = f_i(S) - \tau_i$ as the waiting time of T_i in S. Then it is easily seen that a schedule minimizing $\bar{\omega}$ also minimizes the mean, weighted waiting time.

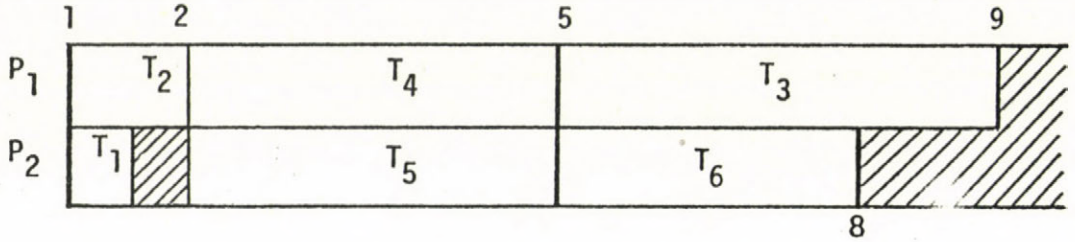
Now suppose the general model is extended to permit a positive number $d_i (1 \le i \le n)$ called the due date to be given for each task T_i. The due date expresses the time at which it is desired to have a task finished. Then the lateness of T_i in S is defined as $f_i(S) - d_i$, and the tardiness is defined as $\max\{0, f_i(S) - d_i\}$. The maxima and weighted means of lateness and tardiness are also interesting performance measures. As with waiting times, it is easily seen that sequences mini-



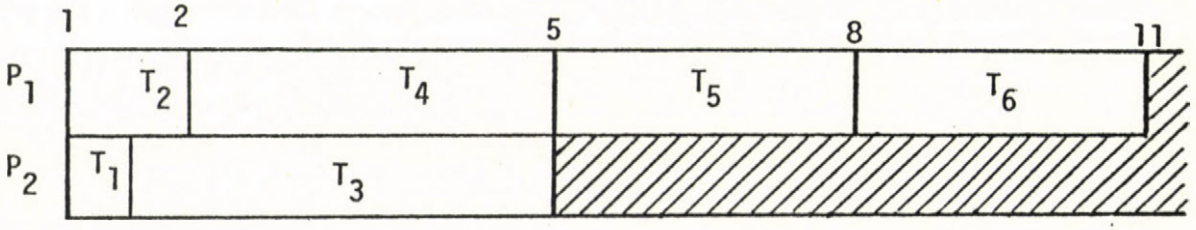
(a)



(b)



(c)



(d)

Figure 3. Discipline hierarchy. (a) $m = 2$ identical processors, $s = 0$, $w_i = 1 (1 \le i \le n = 6)$. (b) Optimal preemptive schedule, $\omega = 8$, $\bar{\omega} = 29/6$. (c) Optimal nonpreemptive schedule, $\omega = 9$, $\bar{\omega} = 30/6$. (d) Optimal list schedule, $\omega = 11$, $\bar{\omega} = 32/6$.

mizing mean weighted flow time also minimize mean weighted lateness; however, this is not true for the mean weighted tardiness.

When we consider problems in which due dates must be respected (i.e., $f_i(S) \leq d_i$, $1 \leq i \leq n$), the due dates are also called deadlines. One such problem to which we devote considerable attention assumes $d_i = d$, $1 \leq i \leq n$, $\prec$ is empty, $s = 0$, identical processors, and seeks to minimize the number of processors required to meet the common deadline d. This problem is referred to as the bin-packing problem. Interestingly, we find that this problem is equivalent to the schedule-length minimization problem for identical processors, $\prec$ empty, $\tau_i = 1$ $(1 \leq i \leq n)$, $m \geq n$, and $s = 1$. Figures 4a and b illustrate this equivalence. Note that if we remove the restriction $m \geq n$ in the parameter list, we have an equivalence to the problem of scheduling to meet a common deadline with the constraint of at most m tasks per processor and the objective of minimizing the number of processors. Although there was no need to introduce deadlines as another component in the general model, the augmented system with an arbitrary set $\{d_i\}$ is discussed briefly in the next section on single-machine results.

III. Background in Single-Processor Results

In this section we cover classical results for the special case of one processor $(m = 1)$, $\prec$ empty, and $s = 0$. For $m = 1$ the problem of minimizing $\max\{f_i\}$ vanishes; hence we are concerned only with the other measures mentioned in the previous section. The important results are summarized in the following theorems.

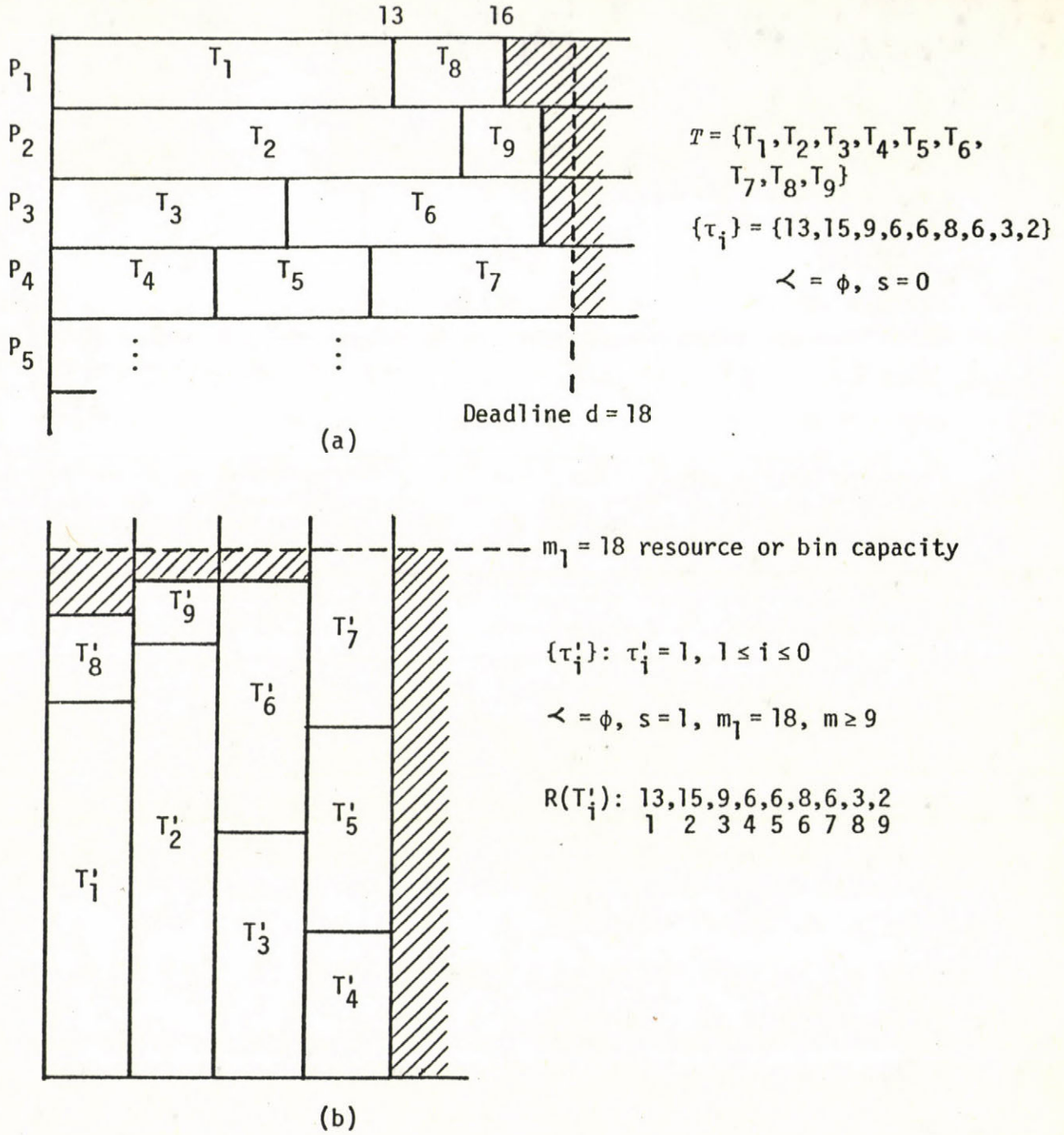


Figure 4. An equivalence of two sequencing problems. (a) Scheduling $\{T_1,\ldots,T_9\}$ to meet deadline d = 18 on minimum number (4) of processors. (b) Scheduling unit-length tasks with single-resource requirements to minimize schedule length.

Theorem 1 [Sm] The mean weighted flow time, lateness, and waiting time are minimized by sequencing the tasks in an order of nondecreasing ratio τ_i/w_i.

Now suppose our system includes a set of due dates for the tasks, and our problem is to optimize performance with respect to the lateness and tardiness measures. We have the following result.

Theorem 2 [Ja2] The maximum lateness and maximum tardiness are minimized by sequencing the tasks in an order of nondecreasing due dates.

Finally, for the due-date problem, let us define the slack time of T_i at time t in a schedule as $d_i - \tau_i - t$. The result of sequencing by the intuitively appealing rule of nondecreasing slack time is given by the following somewhat unexpected result.

Theorem 3 [CMM] Sequencing in an order of nondecreasing slack time maximizes the minimum lateness and minimum tardiness.

Single machine results for problems involving due-dates and the tardiness performance measure have received considerable attention. See [Lal], [HK], and [RLL] (which contains a recent survey) for results related to efficient iterative and enumerative approaches. In a later section we discuss the complexity of these problems.

A well-known related result in [Mo] concerns the problem of finding a sequence that minimizes the number of late tasks. An optimal algorithm can be described as follows. First, the tasks are sequenced in the order of Theorem 2. The algorithm then finds the first late task T in this sequence and eliminates that task T' in the initial subsequence terminated by T, which has the largest execution time. The process is then

iterated on the resulting sequence (with T' removed) until a sequence is obtained with no late tasks. Any sequence beginning with the initial subsequence determined by the above process minimizes the number of late tasks. Extensions have been examined in [Si2].

Another problem using as a performance measure the maximum of the weighted finishing times has been studied in [La2]. A simple algorithm is produced for $\prec$ arbitrary which minimizes $\max\{w_i f_i\}$ on one processor.

It is clearly interesting to assess the importance of sequencing algorithms by comparing their performance with that achieved under a random selection procedure. Moreover, it is also of considerable interest to analyze an algorithm that is optimal under the assumptions of known execution times, under weaker conditions in which only partial information (specifically, a probability distribution) is available. Examination of the difficulties presented by such an analysis of the models in subsequent sections reveals the general intractability of this approach in virtually all cases of interest. However, [CMM] present results for single-processor problems which illustrate the value of probability models.

IV. Results in Schedule-Length and Mean Flow-Time Minimization

We begin with some remarks on measures of complexity. These comments are rather brief, but should be sufficient to permit appreciation of the results described in this section. A careful presentation of these notions appears [U].

In general, the "complexity" of an algorithm solving a given problem refers only to its execution time, expressed as a function of the input-

length; i.e., the number of bits needed to describe an instance of the problem. For our purposes we will usually specify complexity as a function of basic problem parameters, primarily the number n of tasks. In some cases, this is a considerable simplification, but not an inappropriate one for our purposes.

The detailed specification of the functions representing complexity depend on the details of algorithm and data structure design in which we take no special interest. Thus we use the order-of-magnitude notation $O(.)$, which concentrates on the terms of a function that dominate its behavior. Thus if we say that an algorithm has complexity $O(n^2)$, we simply mean that there exists a constant c such that the function cn^2 bounds the execution time as a function of n. As a specific example, if a sequencing rule depends essentially on the ordering of an arbitrary permutation of n execution times, we know that algorithms exist, using binary comparison operations only, whose complexity is $O(n \log_2 n)$. But the specific functions of execution time, for different algorithms, usually consist also of terms $O(n)$, $O(\log_2 n)$, or $O(1)$ and may differ in the coefficient of the $n \log_2 n$ term.

Any sequencing algorithm whose complexity is bounded by a polynomial in n is called a polynomial-time algorithm or an algorithm that runs in polynomial time. The corresponding problem is said to have a polynomial-time solution (algorithm). Henceforth we consider an algorithm to be <u>efficient</u> if it runs in polynomial time. The practical motivations of this terminology are strong, especially for large n. In this respect we should note that the sequencing problems we consider are or can be reduced to finite problems, in the sense that the solution space is finite.

Thus our notion of efficient algorithm can be associated with nonenumerative algorithms; "inefficient" algorithms are those which effectively require a search (enumeration) of the solution space and have a complexity that is at least some exponential in n.

There exists a set of problems, each of which is called NP-complete (sometimes called polynomial complete, or simply complete), which includes many classical, hard problems, Such problems include the traveling salesman problem, finding the chromatic number of a graph, and the knapsack problem, just to mention a few. It is known that in terms of complexity, all the NP-complete problems are equivalent in the following intuitive sense. If one can find a polynomial-time algorithm to solve one of these problems, one can find a polynomial-time algorithm for every other problem in the class, according to a procedure that normally varies from one problem to another. Thus either there exist polynomial-time algorithms for all the NP-complete problems, or none of them has a polynomial-time algorithm; we do not know which of these two assertions is the correct one. However, despite the lack of an answer to this "ultimate" question, there is strong evidence to suggest that all NP-complete problems are inherently intractable. As we shall see, almost all sequencing problems stated in complete generality are NP-complete.

The above intuitive discussion of course is very informal. More formally, one must proceed by defining a common mathematical basis for representing algorithms and instances of problems, defining problem complexity in terms of the model, and defining the transformation (or reducibility) of one problem to another. With such a formalism one can address for specific problems such questions as deciding their complexity

in terms of desirable problem parameters, the tradeoff between storage and execution-time complexity, the essential aspect of a problem that contributes to its complexity, and so on. Later we shall illustrate the techniques for taking a new combinatorial problem and showing that it admits of a polynomial-time solution only if some known NP-complete problem has such a solution.

Schedule-Length Minimization Problems. Table 1 summarizes the majority of results presently known for these problems. To help interpret the table, the following remarks are in order.

1. The columns correspond to the possible parameters of an algorithm that solves a problem defined by the assumptions given in a row of the table. Those entries in which a value is specified eliminate a "free" parameter. For example, in problem (row) 2 we find that m is not a parameter but is fixed at $m = 2$. Similarly, $\{\tau_i\}$ is not a parameter, for the specific common value of the τ_i does not influence the algorithm. The partial order is a free parameter, but no preemptions are allowed and there is no additional resource requirement that can be specified. Here and in the sequel, n is always a free parameter.

2. The entries in the column on complexity given as "open" simply mean that the question of the complexity of the corresponding problem has not been resolved.

3. For each of the nonpreemptive problems for which polynomial-time optimal algorithms are known, there in fact exists an optimal list-scheduling algorithm.

4. An important point concerning the NP-complete problems indicated in Table 1 is that they represent the simplest cases for which NP-completeness is known. Thus the reader can and should make appropriate inferences

regarding more general problems. Generalizing any one of the parameter restrictions in a given problem (e.g., assuming as appropriate, non-identical processors, nonempty partial orders, additional resources, etc.) obviously produces a problem at least as hard as the original one.

One important observation in this respect concerns a comparison of rows such as 6 and 4. Since problem 6 is NP-complete, it is easy to see that so also is the problem for m a free parameter. However, the converse is not necessarily true. Problem 4 is NP-complete, but it is not known whether for any fixed $m \geq 3$ the corresponding problem (problem 3) is NP-complete (and in fact a considerable effort has been invested in resolving this problem for $m = 3$).

5. Regarding polynomial-time algorithms, we do not in all cases claim that the complexity shown is minimal. As an analysis of the sequencing problems indicates, however, there is virtually conclusive evidence to this fact in a number of cases. For example, algorithms necessarily depending on an ordering of execution times must surely be of at least $O(n \log_2 n)$ complexity, assuming an unordered list to begin with. Algorithms depending on precedence constraint structure must surely have a complexity at least $O(n^2)$, since there are $O(n^2)$ edges in a general precedence graph (see problem 2, e.g.). However, complexity must also depend on data structures. For example, if a graph is specified by an edge list containing redundant transitive edges, the complexity may well increase. Problem 2 is a relevant example, for the complexity shown depends on the absence of transitive edges; the problem of removing such edges from a dag edge list has $O(n^{2.8})$ complexity according to the best algorithm currently known [AGU]. Thus without the assumption of the absence of transitive edges, problem 2 would have a complexity dominated by this latter algorithm.

Table 1. Results for Minimizing $\omega = \max\{f_i\}$

	Problem complexity	m	$\{\tau_i\}$*	$\prec$	Rule†	Resources s	Resources $\{m_i\}$	References
1.	$O(n)$	-	Equal	Forest	Nonpr	0		[H]
2.	$O(n^2)$	2	Equal	-	Nonpr	0		[CG]
3.	Open	Fixed $m \geq 3$	Equal	-	Nonpr	0		
4.	NP-complete	-	Equal	-	Nonpr	0		[U]
5.	NP-complete	Fixed $m \geq 2$	$\tau_i = 1$ or 2 for all i	-	Nonpr	0		[U]
6.	NP-complete	Fixed $m \geq 2$	-	ϕ	Nonpr	0		
7.	$O(n \log_2 n)$	-	-	Forest	Pr	0		[MC]
8.	$O(n^2)$	2	-	-	Pr	0		[MC]
9.	Open	Fixed $m \geq 3$	-	-	Pr	0		
10.	NP-complete	-	-	-	Pr	0		[U]
11.	$O(n^3)$	2	Equal	ϕ	Nonpr	-	-	[GJ1]
12.	NP-complete	Fixed $m \geq 2$	Equal	Forest	Nonpr	1	-	[GJ1]
13.	NP-complete	Fixed $m \geq 2$	Equal	-	Nonpr	1	$m_1 = 1$	[U]
14.	NP-complete	Fixed $m \geq 3$	Equal	ϕ	Nonpr	1	-	[GJ1]
15.	$O(n \log_2 n)$	2	Flow shop		Nonpr	0		[Jo]
16.	NP-complete	Fixed $m \geq 3$	Flow shop		Nonpr	0		[GJS]
17.	NP-complete	Fixed $m \geq 2$	Job shop		Nonpr	0		[GJS]
18.	NP-complete	Fixed $m \geq 2$	-	ϕ	Σf_i is minimum	0		[CS]

*Identical processors assumed throughout, except for problems 15-17.

†Nonpr and pr are abbreviations for nonpreemptive and preemptive, respectively.

The optimization algorithms for problems 1, 2, 7, and 8 are essentially critical path algorithms (also called level-by-level algorithms), since the scheduling priority at any point is given sequentially to the remaining tasks at the highest level. (Actually, in problem 2 this criterion must be augmented by another property of tasks which, in the sense of the precedence graph, is locally computable.) Clearly, the complexity of these algorithms is dominated by the complexity of the graph operations needed to find critical paths. Note that "first-order" generalizations of these problems are NP-complete as shown in problems 4, 5 and 10. For special cases applicable to the case of two non-identical processors, see [B2].

Problem 15 is solved essentially by an appropriate ordering of the tasks; hence the complexity is determined by the complexity of sorting. This is the earliest result (1954) in schedule-length minimization. Apart from problem 1 (1961) and special cases of problem 16, the remaining results date from 1968.

The flow-shop problem referred to in problems 15 and 16 is defined as follows. Each task system consists of a set of n/m chains of length m, usually called jobs in the literature, with the restriction that the ith task in a chain must be executed on processor P_i (n is a multiple of m). In terms of $[\tau_{ij}]$ in our original model, columns $km+i$, $k = 0,1,\ldots,n/m-1$, correspond to tasks that must execute on P_i, so that $\tau_{j,km+i} = \infty$ for all $j \neq i$. The T_{km+i}, $i = 1,2,\ldots,m$, will correspond to the sequence of tasks in job k.

Problem 17 concerns the complexity of the simple job-shop problem, described as follows. As in the flow-shop problem $\prec$ is a set of chains, each of arbitrary length, called jobs. (In the general job-shop problem

there is no constraint on $\prec$.) Each job J_i can be characterized by a sequence of pairs (a_{ij},P_{ij}), where a_{ij} is the execution time of the jth task of J_i and P_{ij} is the processor on which it must execute. The general problem is NP-complete for all $m \geq 2$. (Later in this section we illustrate a technique that can be used for this result.) However, there is an interesting special case, which we now describe, that admits of a polynomial-time solution for $m = 2$ [Jal].

Each job J_i is characterized simply by a single operation on each machine (i.e., each J_i is a two-task chain): each J_i either executes a task first on P_1 then on P_2 or it executes a task first on P_2 then on P_1. Let (x_i,y_i) be the execution times for the tasks of J_i. Construct two ordered sets C_1 and C_2 such that all J_i whose first task executes on P_1 (respectively P_2) are elements of C_1 (respectively C_2) and such that if (x_i,y_i) and (x_j,y_j) are both in C_1, then $\min(x_i,y_i) < \min(x_j,y_j)$ implies that J_i precedes J_j in the ordering. (This is the rule used in problem 15.) Order C_2 in the same way. According to this ordering, assign tasks on P_1 first from C_1 then C_2, observing precedence constraints. Similarly, assign tasks on P_2 first from C_2 then C_1. Note that x_i or y_i can be zero, in which case we effectively account for jobs having only a single task for one machine or the other.

The basic intractability of problems in which there are additional resources is made clear in problems 12 to 14. Problem 11, not discussed subsequently, can be viewed as an application of the maximum matching [E] problem. One may construct in less than $O(n^3)$ time an undirected

graph G such that $(T_i,T_j) \in G$ if and only if T_i and T_j are compatible: that is, $R_k(T_i) + R_k(T_j) \leq m_k$, for all k; hence T_i and T_j can execute in parallel. A maximum matching of G provides a shortest-length schedule and can be found in $O(n^3)$ time.

Problem 7 with $\prec = \phi$ has a well-known $O(n)$ algorithm [Mc], which we reconsider later.

We conclude this subsection with a brief description of recent results bearing on deadline problems relevant to the problems we have been discussing. Consider the problem of scheduling equal-execution-time tasks on two identical processors when there is an arbitrary partial order and each task has an individual deadline to meet. If the partial order is given in transitively closed form, it is known [GJ2] how to determine in time $O(n^2)$ whether a valid schedule exists which meets all the deadlines, and if so, generate one. If it is desired to know whether a valid schedule exists that violates at most k deadlines, where k is fixed, this can be done in time $O(n^{k+2})$. However, the problem of the existence of a valid schedule that violates at most k deadlines, k variable, is NP-complete, even for only one processor.

Mean Flow-Time Results. We now present recent results generalizing those in the previous section for single-machine systems. Additional resources are not included because corresponding results do not exist. Discipline constraints are also not considered, since nonpreemptive scheduling is assumed throughout for the same reason. (Recall, however, that a preemptive capability does not accomplish anything more in the case of identical processors, non-negative weights, and an empty partial order.) Comments 1, 2, 4, and 5 describing Table 1 also apply to Table 2 in which we summarize most of the results.

Table 2. Results for Minimizing $\Sigma_i w_i f_i$

Problem complexity	Parameters: n and $\prec$	$\{w_i\}$	$\{\tau_{ij}\}$	m	References
1. $O(n^3)$	ϕ	Equal	-	-	[BCS2,Ho2]
2. NP-complete	Set of chains	Equal	Identical processors	-	[BSe]
3. $O(n^2)$	Forest	-	Identical processors	1	[Hol](See also [Ga,Sil])
4. Open	-	-	Identical processors	1	
5. NP-complete	ϕ	-	Identical processors	Fixed $m \geq 2$	[BCS1]
6. NP-complete	Flow shop			Fixed $m \geq 2$	[GJS]

For problem 1 of Table 2 the existence of a polynomial-time algorithm is indicated for the most general case of independent tasks and equal deferral costs. Special cases of this result have been known for some time, in particular the result for SPT sequencing introduced earlier. The result of problem 1 is based on a formulation that identifies it as a special case of the general transportation problem.

The solution of problem 3 results from a general formulation that proceeds by identifying a locally computable cost function (sequencing criterion) to be associated with each vertex (subtree) of the forest. This function is in fact a generalization of the function τ_i/w_i introduced in Theorem 1. Using an ordering obtained from these computed costs, the optimal sequence is achieved by recursively identifying the subtrees of the forest that are to be executed next.

Problem 6 of Table 2 concerns a flow-shop problem that has received considerable attention. This NP-complete problem forms the basis of an illustration in [KS] of enumerative and approximate techniques applied to sequencing problems.

The problem of minimizing mean tardiness (see previous section) is also relevant to Table 2, with due dates also a parameter. It has been shown that the minimum mean weighted tardiness problem is NP-complete for all $m \geq 1$ [BLR]. An interesting open problem is the complexity of the minimum mean tardiness problem (equal weights) when $m = 1$.

Further Remarks on the Complexity of Sequencing Problems. In [U] a formal approach to the definition of problem complexity is introduced and used in demonstrations of the NP-completeness of a large variety of sequencing problems, either explicitly or implicitly. Such an approach begins with the description of a computer model by which algorithms for combinatorial problems can be commonly posed, in a manner that simplifies their analysis and comparison with respect to their complexity. The notion of polynomial-time nondeterministic algorithms is defined, a notion which identifies with problems admitting of an enumerative solution describable by a polynomial-depth search tree. As a matter of definition, the existence of such an algorithm is necessary for the NP-completeness of a sequencing problem. Finally, the concept of polynomial reducibility among combinatorial problems is defined, a definition used in deciding that a problem has a polynomial-time solution if and only if some NP-complete problem has such a solution.

In connection with Table 1, comment 4 is worth repeating at this point. Namely, the reader will be able to infer the NP-completeness of

new problems from similar results shown for other problems. For example, the NP-complete problem 6 of Table 1 is known to have a polynomial-time solution if and only if a similar solution exists for the classical deadline problem [U]; that is, given a deadline d common to all (independent) tasks, is there a schedule on m identical processors such that all tasks finish no later than d? The consequent NP-completeness of this problem in effect accounts in a straightforward manner for the NP-completeness of the general additional-resource ($s \geq 1$) and bin-packing problems [G3].

We can illustrate informally but effectively the process of reducing one combinatorial problem to another by the following examples. Consider as a basis the simple version of the knapsack problem, stated as follows. Given as parameters a set $X = \{a_1, \ldots, a_n\}$ of n positive integers and an additional integer b, is there a subset of X whose elements sum exactly to b? That is, does the equation $\sum_{i=1}^{n} c_i a_i = b$ have a 0-1 solution in the c_i's? This problem is well known to be NP-complete [Ka]. We now make use of this problem in assessing the complexity of the following sequencing problems.

1. Consider problem 7 of Table 1 with $\prec = \phi$, for which it is readily verified that in general there can be $O(n!)$ solutions. Suppose we add in the "Discipline" column that the number of preemptions must be minimized. Call this the PM(m) problem for m processors.

An example of the case for $m = 2$ and $\max\{\tau_i\} < \sum_{i=1}^{n} \tau_i / 2$ is shown in Fig. 5a. Note that any sequence of tasks can be used in carrying out the assignment; one simply must insert a preemption when necessary for the last task scheduled on P_1 (first task scheduled on P_2). Thus at

most one preemption is necessary, and its necessity depends on finding a subset of $\{\tau_i\}$ that adds up to exactly $\sum_{i=1}^{n}\tau_i/2$. Therefore, we can reduce a knapsack problem to a PM(2) problem as follows. Let b and $\{a_i\}_{i=1}^{n}$ be an instance of the knapsack problem. Consider the instance of the PM(2) problem given by $\{\tau_i\}_{i=1}^{n+1}$ where $\tau_i = a_i$, $1 \le i \le n$, and $\tau_{n+1} = |2b - \sum_{i=1}^{n} a_i|$. It is easy to see that this PM(2) problem has a solution with no preemptions if and only if the original knapsack problem has a solution. Figure 5b provides examples. Clearly, because of the simplicity of the reduction (calculating $|2b - \sum_{i=1}^{n} a_i|$), we expect the PM(2) problem to be at least as hard as the knapsack problem. We make the same statement for the general case $m \ge 3$, since we can add $m-2$ tasks of length $\sum_{i=1}^{n}\tau_i/2$ to reduce the PM(2) problem to the PM(m) problem.

2. Consider problem 16 of Table 1, and let the jobs (i.e., three-task chains) be specified by the triples (x_i,y_i,z_i), $1 \le i \le n$, where x_i (respectively, y_i and z_i) is the time required for the first (respectively, second and third) task to execute on P_1 (respectively, P_2 and P_3). We can provide as follows the basic observation in a proof of NP-completeness.

From an instance b, $\{a_i\}$ of the knapsack problem, let the execution times in an instance of problem 16 for $m = 3$ be given by $(0,a_1,0),\ldots,(0,a_n,0)$, $(b,1,\sum_{i=1}^{n} a_i - b)$. Figure 6 gives an example schedule. Note that a minimal-length schedule is at least $\sum_{i=1}^{n} a_i + 1$ long. But as illustrated, we can know whether we have a minimal-length schedule only if we can answer the question: is there a subset of the a_i's which sum exactly to b? Thus by a simple algorithm we can transform an instance of the knapsack problem to an instance of the $(m = 3)$

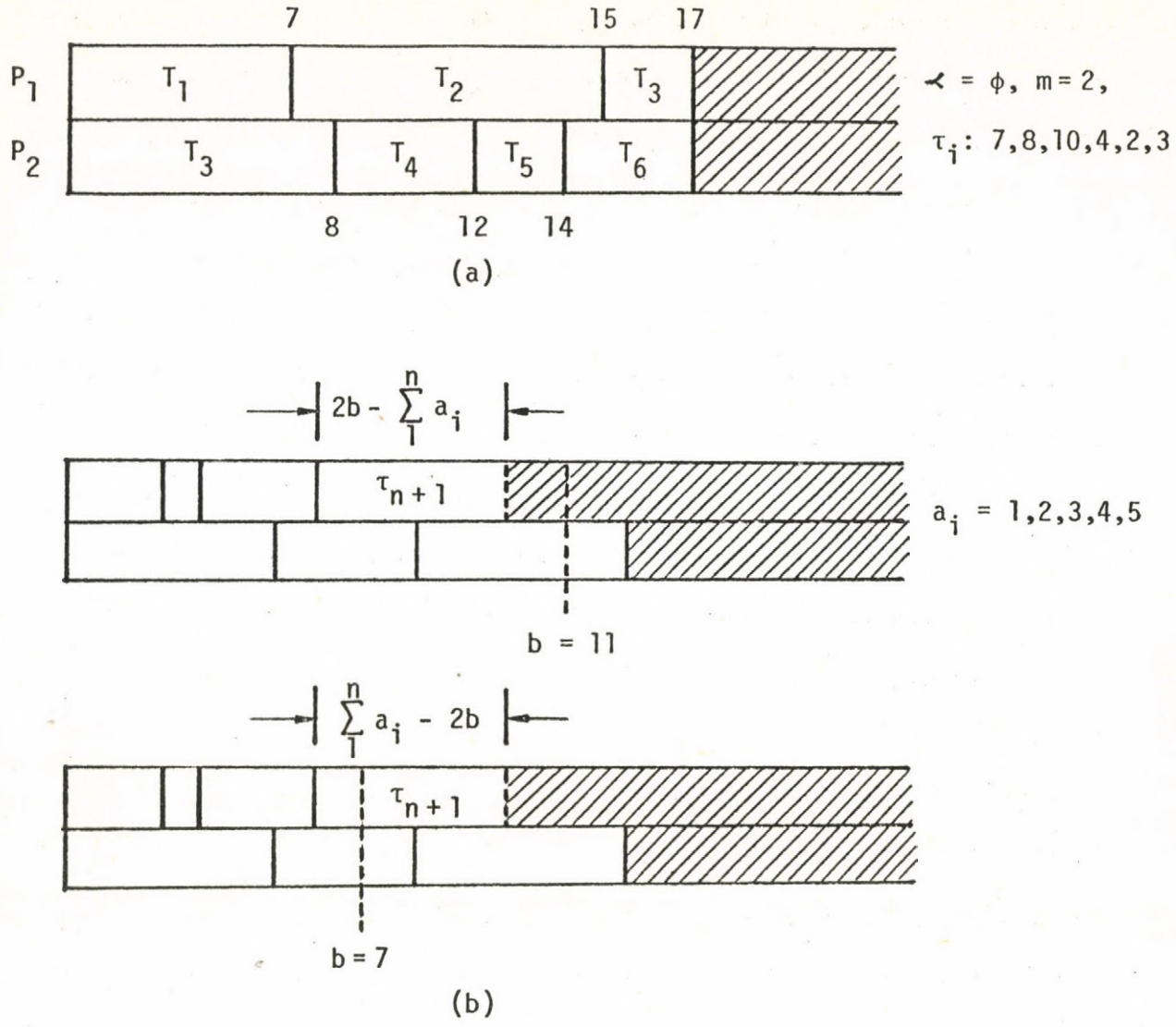


Figure 5. Preemptive scheduling

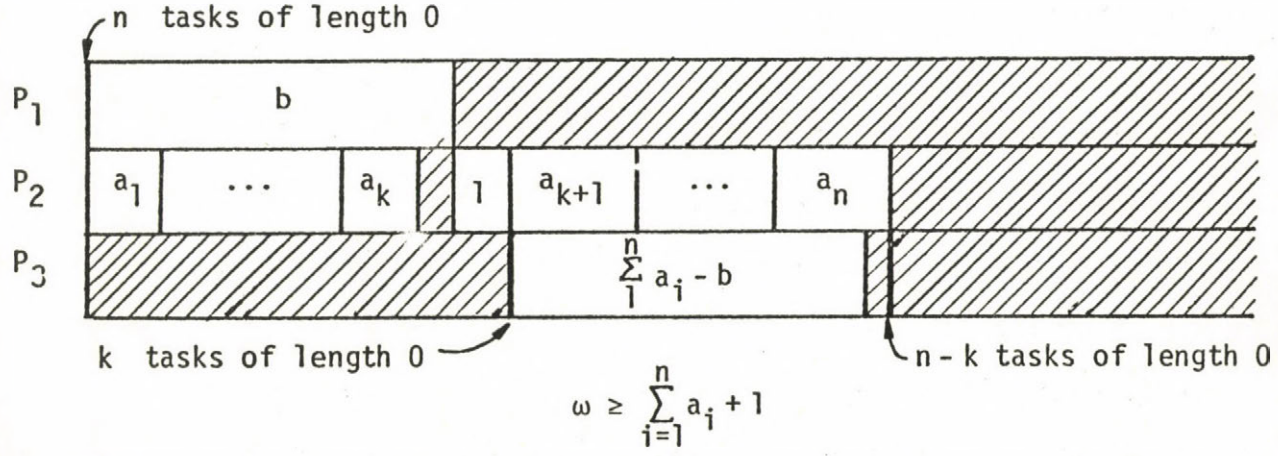


Figure 6. A flow-shop schedule, m = 3

flow-shop problem such that an algorithm for the flow-shop problem finds a minimal solution if and only if the original knapsack problem has a solution. As before, it is not difficult to render a general case for $m > 3$ identical to a problem for $m = 3$. It follows that we can expect the problem to be at least as hard as the knapsack problem.

A similar type of argument can be easily constructed for the result of problem 17, taking advantage of the absence of the "processor ordering" implicit in the flow-shop problem. It is important to note that the knapsack problem can be solved in time polynomial in the desired sum b (this does not contradict the NP-completeness of the knapsack problem since b can be expressed using $\log_2 b$ bits). Therefore, if the complexity parameter of interest is the sum of task execution times we can not infer from the above arguments that these problems are intractable. For example, Problem 6 in Table 1 can be solved in polynomial time for fixed $m \geq 2$ for this less stringent input measure. However, Problems 16 and 17 remain NP-complete even for the latter measure [GJS].

Clearly, we have only shown these problems to be at least as hard as NP-complete problems; we have not verified that they are no harder. At this point it is best to leave full proofs of NP-completeness to those familiar with [U]. We might note here, however, that the problem of showing reducibility has no generally applicable structure (thus the open problems of Tables 1 and 2), and solutions to these problems can be quite involved.

We conclude this discussion of complexity with the warning that having a specific problem in hand, one should be fully aware of any special features or constraints that might exist. For in this case the

NP-completeness of the general problem to which it corresponds should not immediately cause us to abandon hope for an efficient optimal algorithm. Specific limitations on the problem may enable us to design an algorithm whose execution time is bounded by a polynomial in the basic characteristics of the problem.

V. Concluding Remarks

We have limited our discussion to efficient optimization algorithms and to problem complexity. The results on complexity demonstrate rather clearly that much of the progress in the theory of scheduling is very likely to result from the study of fast heuristics (approximate algorithms) and effective enumerative procedures. In [G2] and [G3] performance bounds are presented and derived for a variety of sequencing and allocation heuristics whose complexity is at most $O(n^2)$. We refer to [KS], [CMM] and [Ba] for extensive treatments of enumerative techniques including dynamic programming and local neighborhood search.

One performance measure of broad interest that we have neglected, and for which results have been obtained very recently, is the mean number of tasks, $\bar{N}(S)$, in the system calculated over the interval $[0,\omega(S)]$. In general, such a measure is useful in expressing expected inventory or storage requirements for tasks in the job shop or computer. We can write

$$\bar{N}(S) = \frac{1}{\omega(S)} \int_0^{\omega(S)} N(t)dt$$

where $N(t)$ is the number of uncompleted tasks in the system at time t. Clearly $N(t)$ takes the form of a decreasing staircase function with

changes in value occurring at task completion times (subsets of which may, of course, be coincident). Assuming unit weights in (2), one may show rather easily

$$\bar{N}(S) = n \frac{\bar{\omega}(S)}{\omega(S)}$$

which is obtained independently of scheduling disciplines, number of processors, or any property other than finishing times. From this expression we note immediately that for $m=1$ processors $\bar{N}(S)$ is minimized when $\bar{\omega}(S)$ is minimized, since $\omega(S)$ is in fact not a function of S for $m=1$. For the case $m \geq 2$ it is known [CL] that the problem of finding a sequence minimizing $\bar{N}$ is NP-complete.

A recent generalization of our general model for which an efficient optimal algorithm has been found is described as follows. With respect to problem 3 of Table 2 we once again assume a forest and a single processor, but each tree (called a job) in the forest is now regarded as a decision tree in the following sense. There is an arbitrary (discrete) probability distribution associated with each vertex governing the decision as to which task (vertex) is to be executed next in the job. Thus the eventual execution of a job consists of a sequence or chain of tasks beginning with the root and ending at a descendant vertex, and the chain can be interrupted only at task termination times. These sequences and their total execution times are random variables, hence the problem becomes one of minimizing the expected value of the mean weighted flow time. A sequencing criterion for deciding sequentially which job is next to have a task executed has been developed along the lines of problem 3. The complexity question is studied, with the result that an $O(n^2)$ algorithm is produced [BCJ].

References

[AGU] Aho, A. V., M. R. Garey, and J. D. Ullman, "The Transitive Reduction of a Directed Graph," SIAM Journal on Computing, 1 (1972), 131-137.

[B1] Baer, J. L., "A Survey of Some Theoretical Aspects of Multiprocessing," Computing Surveys, 5, 1 (1973).

[B2] Baer, J. L., "Optimal Scheduling on Two Processors of Different Speeds," Computer Architectures and Networks, E. Gelenbe and R. Mahl (Eds.), North Holland Publishing Company, 1974, 27-45.

[Ba] Baker, K., Introduction to Sequencing and Scheduling, John Wiley and Sons, 1974.

[BCJ] Bruno, J., E. G. Coffman, Jr., and D. B. Johnson, "On Batch Scheduling of Jobs with Stochastic Service Times and Cost Structures on a Single Server," Technical Report No. 154, Computer Science Dept., Pennsylvania State University, August 1974, Journal of Computer and System Sciences (to appear).

[BCS1] Bruno, J., E. G. Coffman, Jr., and Ravi Sethi, "Scheduling Independent Tasks to Reduce Mean Finishing Time," Communications of the ACM, 17, 7 (1974), 382-387.

[BCS2] Bruno, J., E. G. Coffman, Jr., and Ravi Sethi, "Algorithms for Minimizing Mean Flow Time," Proceedings, IFIPS Congress, North Holland Publishing Co., August 1974, 504-510.

[BLR] Brucker, P., K. K. Lenstra, and A. H. G. Rinooy Kan, "Complexity of Machine Scheduling Problems," Technical Report BW 43/75, Mathematisch Centrum, Amsterdam, 1975, Operations Research (to appear).

[BSe] Bruno, J., and R. Sethi, "On the Complexity of Mean Flow-Time Scheduling," Technical Report, Computer Science Dept., Pennsylvania State University, 1975.

[C1] Coffman, E. G., Jr., "A Survey of Mathematical Results in Flow-Time Scheduling for Computer Systems," Proceedings, GI 73, Hamburg, Springer-Verlag, 1973, 25-46.

[C2] Coffman, E. G., Jr., (Editor) Computer and Job-Shop Scheduling Theory, John Wiley and Sons, 1975.

[CD] Coffman, E. G., Jr., and P. J. Denning, Operating Systems Theory, Prentice Hall, Englewood Cliffs, N. J., 1973.

[CG] Coffman, E. G., Jr., and R. L. Graham, "Optimal Scheduling for Two Processor Systems," Acta Informatica, 1, 3 (1972), 200-213.

[CL] Coffman, E. G., Jr., and J. Labetoulle, "Deterministic Scheduling to Minimize Mean Number in System," Proceedings, HICCS 9, University of Hawaii (to appear Jan. 1976).

[CMM] Conway, R. W., W. L. Maxwell, and L. W. Miller, Theory of Scheduling, Addison-Wesley, Reading, Mass., 1967.

[CS] Coffman, E. G., Jr., and R. Sethi, "Algorithms Minimizing Mean-Flow-Time: Schedule-Length Properties," Technical Report, Computer Science Dept., Pennsylvania State University, 1973, Acta Informatica (to appear).

[E] Edmonds, J., "Paths, Trees, and Flowers," Canadian Journal of Mathematics, 17 (1965), 449-467.

[G1] Graham, R. L., "Bounds on Multiprocessing Timing Anomalies," SIAM Journal on Applied Math., 17 (1969), 416-429.

[G2] Graham, R. L., (Chapter 5 of [C2]).

[G3] Graham, R. L., "Bounds on Multiprocessing Anomalies and Related Packing Algorithms," Proceedings, AFIPS Conference, 40 (1972), 205-217.

[Ga] Garey, M. R., "Optimal Task Sequencing with Precedence Constraints," Discrete Math., 4 (1973) 37-56.

[GJ1] Garey, M. R. and D. S. Johnson, "Complexity Results for Multiprocessor Scheduling Under Resource Constraints," Proceedings, 8th Annual Princeton Conference on Information Sciences and Systems, 1974.

[GJ2] Garey, M. R. and D. S. Johnson, "Deadline Scheduling of Equal Execution Time Tasks on Two Processors," Technical Report, Bell Laboratories, Murray Hill, N. J., 1975.

[GJS] Garey, M. R., D. S. Johnson, and R. Sethi, "The Complexity of Flow Shop and Job Shop Scheduling," Technical Report No. 168, Computer Science Dept., The Pennsylvania State University, 1975.

[H] Hu, T. C., "Parallel Sequencing and Assembly Line Problems," Operations Research 9, 6 (1961), 841-848.

[HK] Held, M. and R. Karp, "A Dynamic Programming Approach to Sequencing Problems," SIAM Journal on Applied Math., 10, 11 (1962), 196-210.

[Ho1] Horn, W. A., "Single-Machine Job Sequencing with Treelike Precedence Ordering and Linear Delay Penalties," SIAM Journal on Applied Math., 23 (1972) 189-202.

[Ho2] Horn, W. A., "Minimizing Average Flow Time with Parallel Machines Operations Research, 21 (1973), 846-847.

[Ja1] Jackson, J. R., "An Extension of Johnson's Results on Job-Lot Scheduling," Naval Research and Logistics Quarterly, 3, 3 (1956).

[Ja2] Jackson, J. R., "Scheduling a Production Line to Minimize Maximum Tardiness," Research Report No. 43, Management Sciences Research Project, UCLA, January 1955.

[Jo] Johnson, S. M., "Optimal Two- and Three-Stage Production Schedules," Naval Research and Logistics Quarterly, 1, 1 (1954).

[Ka] Karp, R. M., "Reducibility Among Combinatorial Problems," Complexity of Computer Computation, R. E. Miller and J. W. Thatcher (Eds.), Plenum Press, New York, 1972, 85-104.

[KS] Kohler, W. H. and K. Steiglitz (Chapter 6 in [C2].)

[La1] Lawler, E. L., "On Scheduling Problems with Deferral Costs," Management Science, 11 (1964), 280-288.

[La2] Lawler, E. L., "Optimal Sequencing of a Single Machine Subject to Precedence Constraints," Management Science, 14 (1973), 544-546.

[Mc] McNaughton, R., "Scheduling with Deadlines and Loss Functions," Management Science, 12, 7 (1959).

[MC1] Muntz, R. R. and E. G. Coffman, Jr., "Preemptive Scheduling of Real Time Tasks on Multiprocessor Systems," Journal of the ACM, 17, 2 (1970), 324-338.

[MC2] Muntz, R. R. and E. G. Coffman, Jr., "Optimal Preemptive Scheduling on Two-Processor Systems," IEEE Transactions on Computers, C-18, 11 (1969), 1014-1020.

[Mo] Moore, J. M., "An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs," Management Science, 15, (1968), 102-109.

[RLL] Rinnooy Kan, A. H. G., B. J. Lageweg, and J. K. Lenstra, "Minimizing Total Costs in One-Machine Scheduling," Technical Report No. BW33/74, Mathomatisch Centrum, Amsterdam, 1974.

[Si1] Sidney, J. B., "One Machine Sequencing with Precedence Relations and Deferral Costs - Part I.," Working Paper No. 124, "Part II," Working Paper No. 125, Faculty of Commerce and Business Administration, University of British Columbia, 1972.

[Si2] Sidney, J. B., "An Extension of Moore's Due-Date Algorithm," Symp. on Theory of Scheduling and Its Applications, S. M. Elmagrabhy (Ed.), Springer-Verlag, 1973, 393-398.

[Sm] Smith, W. E., "Various Optimizers for Single-Stage Production," Naval Research and Logistics Quarterly, 3, 1 (1956).

[U] Ullman, J. D., (Chapter 4 in [C2].)

Összefoglaló

Determinisztikus ütemezés: "komplexitás" és optimális algoritmusok

Coffman E. G.

A dolgozat korlátokat ad a hatékonyságra különféle ütemezések esetén, ha a "komplexitás" legfeljebb $O(n^2)$. Optimális ütemezési algoritmusokat ad preemptiv és nempreemptiv esetekben a részben rendezett "task"- rendszerek számára. Részletesen elemzi a "polinomálisan teljes" feladatok esetét.

Р Е З Ю М Е

ДЕТЕРМИНИЧЕСКОЕ РАСПИСАНИЕ: "СЛОЖНОСТЬ" И ОПТИМАЛЬНЫЕ АЛГОРИТМЫ

Коффман Э.Г.

В работе даны ограничения для эффективности, в случае разных расписаний, если сложность имеет порядок $O(n^2)$. Даются оптимальные алгоритмы расписания в "преэмптивном" и непреэмтивном" случаях для упорядоченных систем тасков.

Подробно изучается случай полиномиально полной задачи.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények, 18/1977.

PROGRAM BEHAVIOR IN MULTIPROGRAMMED COMPUTERS AND DIFFUSION APPROXIMATION

Mátyás Arató

One of the most important concept in analysis of computer program behavior is the *locality* property. In operating systems of computers the memory management problem – i.e., deciding how to distribute information among the levels and when to more it about – has of first importance. The virtual memory techniques for memory management are becoming widerspread. A virtual memory can be regarded as the main memory of a simulated computer. It is described in terms of the following abstractions: address space, memory space, and an address map. The address spece of a job is the set of addresses that can be generated by a processor as it executes the job. If the blocks, which are the units of allocation and transfer of information, are the same uniform size they are called *pages*. Most of the results about memory management are stated alwasy in terms of paging. A job's *reference string* $\eta_1, \eta_2, \ldots, \eta_t$, is defined such that η_t is the number of the page containing address x_t, $t \geqslant 1$. The reference string is used as the basis for the models of program behaviour. A reference string of pages satisfies the locality property if, during any interval of time, the program's pages are referenced nonuniformly, and the page reference densities change slowly in time.

One may define the *working set* of a program to be the smallest subset of its pages that must be in main memory at any given time in order to guarentee the program a specified level of processing efficiency. The *working set principle* of memory management asserts that a program (or job) may be designated as processable only if its working set is present in main memory. These definitions and practical measuring techniques were proposed by many authors (see e.g. Denning (1968) and Coffman, Denning's book (1974)).

Under the assumption of locality it is possible to demonstrate that the size of a job's working set tends to be normally distributed. This is a significant result, for it means that the theory of Gaussian processes can be applied to the study of memory management. This result is due to Denning and Schwartz (1972) and agrees with the experiments (see Coffman and Ryan (1972)).

When dealing with a large program it is impossible, in practice to predict the references deterministically, and it has been recognized that one has to resort to probabilistic models in this context. The choice of the correct probabilistic model is far from obvious. Here we want to give an improved model to understand the stochastic structure underlying the phenomene. Here we do not suggest methods for improvements in the decision algorithms (for such purposes we send the reader to Arató [1], [2] and Benczúr-Krámli-Pergel).

A program's working set $W(t,T)$ at time t is defined to be the set of distinct pages referenced in the time interval $[t - T + 1, t]$, i.e. among $\eta_{t-T+1}, \eta_{t-T+2}, \ldots \eta_t$. The parameter T is called the *window size,* it can be chosen large enough so that the probability of a currentlocality page's, being missing from the working set is small and, small enough so that the probability of more than one interlocality transition's being contained in the window is small.

A program's reference string $\eta_1, \eta_2, \ldots, \eta_t, \ldots$ we regard as a sequence of random variables. We assume that η_t is a simple Markov chain with stationary transition probabilities.

$$p_{ij}^{(n)} = P\{\eta_{t+n} = j \mid \eta_t = i\},$$

and

$$f_i(n) = \begin{cases} p_{ii}^{(1)}, & n = 1 \\ p_{ii}^{(n)} - \sum_{k=1}^{n-1} f_i(k)\, p_{ii}^{(n-k)}, & n > 1. \end{cases}$$

Further assumptions about reference string are that each page is recurrent, i.e.

$$\sum_{n \geqslant 1} f_i(n) = 1$$

and that η_t and η_{t+n} become uncorrelated in the limit $(n \longrightarrow \infty)$.

The *working set size* $w(t,T)$ is the number of pages in $W(t,T)$. As

(1) $$w(t,T) = w(t-1,T) + \Theta(t,T)$$

where

$$\Theta(t,T) = \begin{cases} -1 \\ 0, \\ 1, \end{cases}$$

according to the assumptions on η_t it can be proved that $w(t,T)$ is asymptotically normally distributed as $t \longrightarrow \infty$ and $T \longrightarrow \infty$. The proof of this statement depends on the "mixing" property of homogeneous stochastic processes (see e.g. M. Rosenblatt, J. Rozanov or I. Ibragimov- J. Linnik).

In most applications it is more easy and simpler to deal with the sequence $w_n = w(t_n,T)$, where $t_0 < t_1 < 0 \ldots < t_n < t_{n+1} < \ldots$ and $t_n - t_{n-1} \geqslant 1$ (large enough).
In this case we may use the normal approximation for w_n and heuristically we get (with $Ew = m_n$, $w_n' = w_n - m_n$)

(2) $$w'_n = \rho w'_{n-1} + \epsilon_n, \qquad |\rho| < 1,$$

with independent sequence ϵ_n. The matter is that time can be measured in at least two ways: either with respect to the flow of instructions, or with respect to new page references. Because the first method is more informative when studying overhead caused by page faults, it is adopted in most cases. The second method is sufficiently information in almost all other cases, e.g. for dynamic partitioning strategy (see Coffman and Ryan (1972)).

Relation (2) means that the probability that w'_n increases at the next step is inversely proportional to w'_n; the process has a tendency to approach the mean that is a function of n.

When we are using the model (1) we cannot assume that $w(t)$ (T is fixed) is a stationary process, even when $\Theta(t,T)$ are not independent. But using the scaling factor $t_n - t_{n-1} = s_n$ and taking

$$w_{t_n} = w_n$$

and then the approximation that n is continuous it may be assumed that w_n is stationary. This last assumption means that we are taking a rared sequence of $w(t,T)$ and then (as $n \longrightarrow \infty$ too) we are looking the process w_n on a new axis, see Fig.1.

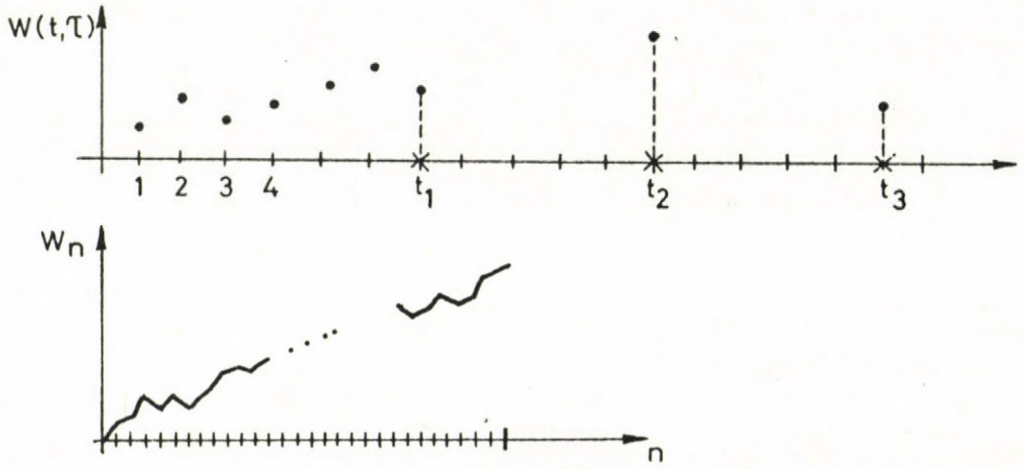


Figure 1.

The first order autoregressive model was used by Coffman and Ryan, where they tested the normal approximation with Monte-Carlo method (simulation). Empirical results also suggested the normal approximation for large "window" sizes, T and larger $t_n - t_{n-1}$. For smaller values of theses parameters the distribution is more closed by Poisson distribution.

When asymptotic uncorrelation assumption is not met for $\Theta(t,T)$ (as $t \to \infty$) the normal approximation to $w(t,T)$ may be poore (see e.g. Rodriguez–Rosell (1971)) Pages are assumed to be of constant size and the entire storega hierachy, main memory and auxiliary storage, in which paging takes place is taken to contain of $\{1,2,\ldots,n\}$ pages, where n is the total number of pages. Let k be the maximum number of pages that can reside in main memory. Under *demand paging*, a page that is referenced but is not resident in main memory will be brought in from auxiliary storage, usually in place of some other page which has to be pushed out, according to the particular paging algorithm implemented in the oparating system.

A common class of such algorithms is the so-called stack algorithm, which we define in the following way (see Denning [2]). The set $D = \{1,2,\ldots,n\}$ denotes the set of pages of a given program, so that the members of the reference string $\eta_t \epsilon D$. The program has been allocated a main memory space of m pages $m \leqslant n$ and $m \leqslant k$. We call subset S of D such that S contains m or fewer pages a possible *memory state*. Let $S(A,k,r)$ denote the memory state after A(the allocation algorithm) has processed r pages under demand paging in an initially empty main memory of size k. The A is a stack algorithm if $S(A,k-1,r)$ is a subset of $S(A,k,r)$. That is the contents of the $(k-1)$ page memory are always contained in the k-page memory, so that the memory states are "stacked up" on one another. The most popular algorithm is LRU, least recently used, because in some Markov models it is optimal. For stack algorithms the performance is crucially dependent on the behavior of the *distance string*, defined as follows. Suppose that at time t page $\eta_t = i$ has been referenced, and that the next reference to page i occurs at time $t + n_t + 1$; that is, between these two references to page i there have been n_t page references, but none to page i. Let d_t denote the number of distinct page references among these $\eta_t$references. The string $(d_1, d_2, \ldots)$ is called the *distance string* corresponding to the reference string η_t. A page exception will occur each time that $d_t = k$. If we assume that (see Freiberger, Grenander, Sampson 1975) at time t a page i is selected from $\{1,2,\ldots,n\}$ according to the probability distribution $p = \{p_1,\ldots,p_n\}$ and then subsequent references are made to this page i a number $\nu_i - 1$ times, where

$$P(\nu_i = k) = (1 - q_i)\, q_i^{k-1}, \quad k = 1,2,\ldots \quad (0 \leqslant q_i \leqslant 1),$$

then the lenght of the reference string (n_t) is not normally distributed. The distribution of the distence d_t under the above conditions, with $p_i = 1/n$ and $q_i = 0$, has the following surprisingly simple form

$$P(d_t = k) = \frac{1}{n},$$

which is far from Gaussian distribution.

Empirical results (see Freiberger, Grenander and Sampson) show that stochastic dependence between some pages must exist contradicting of the above assumptions in the analytic model. The empirical autocorrealation coefficient of order one indicates that the time series of working set size cannot be a white noise, it can be regarded as a first order autoregressive series with coefficient $\rho = 0{,}2 - 0{,}3$.

Spectral tests were used (see Lewis, Shedler 1973) to show the distance strings are correlated. That the $d_t' - s$ are near a first order Markov chain (see Table 1.) they get empirically

Counts of one step transitions for LRU distances							
j/k	1	2	3	4	5	6	7
1	2,943.817	840.912	210.914	78.002	41.500	24.281	16.792
2	1,048.310	2,151.371	146.163	35.192	26.012	13.591	7.931
3	130.957	271.850	176.386	16.570	5.693	2.804	2.318
4	22.878	70.630	35.013	10.338	3.721	1.516	1.643
5	18.512	36.744	16.366	5.258	4.017	393	235
6	10.685	20.180	7.959	1.650	812	1.914	223
7	11.549	11.324	3.596	846	280	233	271
8	7.957	9.934	4.405	1.261	182	128	61
9	9.451	8.248	3.457	834	184	145	56
10	7.274	8.657	2.641	769	639	268	76

Table 1.

From the sequence of address traced, the distance string was derived by stack processing techniques for a page size of 4K (4096) bytes. The data consisted of $t_0 = 8{,}802.464$ references to a total of 517 distinct pages.

For the exception process the results are in Table 2.

	Memory capacity c (pages)		
	76	197	512
N	1807	820	517
$\hat{\rho}_1$ – estimated serial correlation coefficient of order 1 for times between page exceptions	+ 0.188 (0.08)	+ 0.177	+ 0.130
$(N-1)^{\frac{1}{2}} \hat{\rho}_1$	+ 8.01 (1.7)	+ 5.11	+ 3.00
$\hat{\alpha}_{22}$ – estimated partial serial correlation of order 2	0.035 (0.002)	0.065	0.002

Table 2.

From this table 2. one can see that the estimated serial correlation of lag 1, $\hat{\rho}_1$, is too large to be consistent with a true value $\rho_1 = 0$ (the asymptotic variance is $\frac{1}{(N-1)^{1/2}}$, where N is the number of observation). Another outstanding feature is that the estimated partial serial correlations, $\hat{\alpha}_{22}$, are very small, suggesting first-order Markov dependence of intervals between exceptions. The partial correlation of order two is (see Arató. Benczúr, Krámli, Pergel (1976))

$$\alpha_{22} = \frac{\rho_2 - \rho_1^2}{1 - \rho_1^2}$$

In case of first-order Markov dependence $\rho_k = \rho^k$ and $\alpha_{22} = 0$.

At last we give the empirical correlation function of the memory state at the second level on a CDC 3300 computer, the so-called scratch-pool requirement. The scratch-pool requirements of jobs depend on time stochastically and the benchmark measurements were executed during runtime (see Tőke, Tóth, (1975)). The samples are taken in every minutes for 5 hours. The empirical correlation function has an $e^{-\lambda t}$ (t – time) form, which indicates the first order autoregressive scheme, where after 15-20 minutes the correlation is practically equal to 0 (see Fig. 2.).

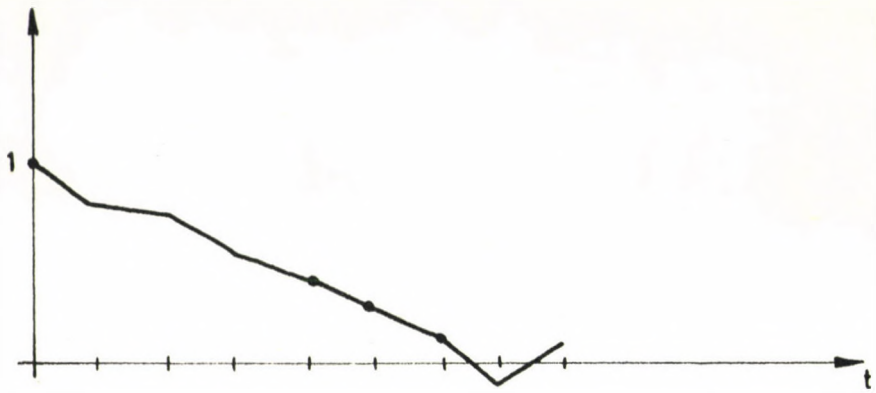


Figure 2.

References

[1] M. Arató, A. Benczúr, A. Krámli, J. Pergel, Statistical problems of the elementary Gaussian processes (1976.) SzTAKI Tanulmányok (1976) 41.

[2] M. Arató, [1] A note on optimal performance of page storage hierarchies Acta Cybernetica (in print) (1976)

[3] M. Arató, [2] Számitógépek hierarchikus laptárolási eljárásainak optimalizálásáról (in Hungarian) (1976) SzTAKI Közlemények 16

[4] A, Benczúr, A. Krámli, J. Pergel, On optimal algorithms of demand paging (1976). Acta Cybernetica (in print)

[5] E. Coffman, P. Denning, Operating Systems theory Prentice-Hall, New Yersey (1973)

[6] E. Coffman, T. Ryan, A study of storage partitioning using a mathematical model of locality (1972) Comm. A.C.M., 185-190.

[7] P. Denning [1] (1968) Resource allocation in multiprocess computer systems. MIT Project MAC report MAC-TR-50 Cambridge, Mass.

[8] P. Denning, [2] (1970) Virtual Memory Computing Surveys 2, 3 (1970).

[9] P. Denning, Schwartz, Properties of the working set model Comm. A.C.M. 191-198 (1972).

[10] W. Freiberger, U. Grenander, P. Sampson, Patterns in page references (1975) IBM J. Res. Dev. 230-243.

[11] I. Ibragimov, Ju. Limik, Independent and stationary related random variables (in Russian) Nauka, Moskwa.

[12] P. Lewis, G. Shedler, Empirically derived micromodels for sequences of page exceptions (1973). IBM J. Res. Dev.

[13] Rodriguez-J. Rosell, Experimental data on how program behaviour affects the choice of scheduler parameters, Proc. 3 rd ACM Symp. on Op. Syst. Princ. (1971)

[14] M. Rosenblatt, A central limit theorem and a stong mixing condition. Proc. Nat. Acad. Sci. USA 42, 43-47. (1956).

[15] Ju. Rozanov, Stationary stochastic processes (in Russian) Fizmatgis, Moskwa, (1963).

[16] P. Tőke, B. Tóth, A scratch-pool utilization study (in Hungarian) SzTAKI Közlemények 15, 103-110. (1975).

Összefoglaló

Multi programozású számitógépek program viselkedése és diffuziós közelitése

Arató Mátyás

A dolgozatban a hivatkozási string sztochasztikus viselkedése alapján vizsgáljuk az u. n. munkamezők nagyságának Gauss eloszlással történő közelitését. Empirikus adatok alapján az időbeli változás egyszerü autoregressziós sémával irható le, amely diszkrét diffúzios folyamat.

Р Е З Ю М Е

Поведение программ в ЭВМ с многопрограмным режимом и диффузионные приближения.

Матяш Арато

В статье рассматриваются вопросы приближения с нормальным распределением так называемых рабочих полей когда поведение " reference string " является случайным. На основе эмпирических данных можно утверждать, что схема авторегрессии является хорошим приближением для описания процесса рабочего поля с дисвретным временем.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

A BAYESIAN APPROACH TO THE PROBLEM OF OPTIMAL PROGRAM–PAGING STRATEGIES

A. Benczur – A. Krámli

Several works in the recent literature on computer sciences are devoted to the optimal program-paging problem (see e.g. Franaszek and Wagner [1], Denning [2]. Starting from the standpoint of these works M. Arató [3] proposed a statistical approach to this problem using Bayesian sequential decision theory developed by Bellman (see. e.g. [4]).

We give the most important special case the exact proof of the optimality of the "least frequently used" strategy

We repeat briefly the model used by Arató, to make more understandable this note.

The program consists of n pages and the computer has a twolevel hierachical memory. We suppose that k pages can be stored in the high speed memory (central memory), and $n - k$ pages (in fact the whole program) is stored on a slower access memory device (disk).

The sequence of references to the different pages forms a sequence $\{\eta_t\}$ of i.d.d. random variables with unknown probability distribution

$$P_i = P(\eta_t = i) \qquad i = 1,2, \ldots, n.$$

In Arató's model the pages can be rearranged before each reference without extra cost. (Such a situation is realized if the pages are subroutines of a main program, and between two consecutive calls there is a relatively great time interval.) But if the main program calls a routin, which is absent from the central memory the cost increases with 1 unity.

As in the case of known distribution $\{P_i\}$ the optimal strategy depends only on the order of probabilities P_i (before every call we must store the first k most probable pages in the central memory) we suppose, that the unkown parameter of the distribution is the permutation w of n elements giving the right order of P_i-s in the following way:

If $\{P_1, \ldots, P_n\}$ is a decreasing sequence of probabilities $(P_1 + \ldots + P_n = 1)$, then $P_i = p_{w(i)}$ where $w(i)$ denotes the one to one mapping realized by the permutation w.

By the Bayesian principle we suppose that the parameter w is also a random variable. If we have no previous information about its distribution, we can consider every permutation equally like, i.e. the apriori distribution ξ of n is the uniform distribution on $n!$ elements. In Arató's work is treated the case of arbitrary apriori distribution. Let us denote by D_N the set of sequential decision procedures $\{d_0, \ldots, d_{N-1}\}$ for a program consisting of N step. By Arató's model d_t is a subset of the labels of the pages, wich are absent from the central memory after the observation of references

$$\eta_1, \ldots, \eta_t.$$

Before the program is started we define d_0 on the basis of the a priori distribution ξ; by the symmetriry of ξ, d_0 can be choosen arbitrarily.
Let us introduce the random process $x_t^{d_{t-1}}$

$$X_t^{d_{t-1}} = \begin{cases} 0 & \text{if} \quad \eta_t \notin d_{t-1} \\ 1 & \text{otherwise} \end{cases}$$

By this terms the utility function has the form:

$$(1) \qquad V(\xi, N) = \max_{\{d_1, \ldots, d_{N-1}\} \in D_N} E_\xi \left(\sum_{t=1}^{N} X_t^{d_{t-1}} \right)$$

E_ξ means the expectation taken on the basis of the uniform a priori distribution ξ. Our aim is to determine the class $D'_N \subset D_N$ of sequential decision procedures for which the maximum in (1) is reached. The solution of this problem can be obtained by solving recursively the Bellman equation

$$(2) \qquad V(\xi(\eta_1, \ldots, \eta_{t-1}), N - t + 1) = \max_{d_{t-1}} E_{\xi(\eta_1, \ldots, \eta_{t-1})}(X_t^{d_{t-1}}) + V(\xi(\eta_1, \ldots \eta_t), N - t),$$

where $\xi(\eta_1, \ldots, \eta_t)$ means the aposteriori distribution of the parameter w after having observed the references $\eta_1, \ldots, \eta_t$.

As the aposteriori distribution does not depend on the decision procedure the right hand side of (2) has maximum for such a $d_t - 1$ which maximize $E_{\xi(\eta_1, \ldots, \eta_{t-1})}(X_t^{d_{t-1}})$.

We shall prove the following

Lemma 1. *The expectation* $E_{\xi(\eta_1, \ldots, \eta_{t-1})}(X_t^{d_{t-1}})$ *reaches its maximum for the decision* d_{t-1} *consisting of the* $n - k$ *least frequently used pages.*

If there are pages of equal frequencies their changing by each other has no influence on the expectation

$$E_{\xi(\eta_1, \ldots, \eta_{t-1})}(X_t^{d_{t-1}})$$

The proof of the Lemma can be carried out by direct calculation and comparison of the aposteriori probabilities.

First we calculation the aposteriori probabilities of a fixed permutation w. If we denote by W the set of all permutations of natural numbers $1 \ldots n$, and by $\{k_1, \ldots, k_n\}$ the frequencies of pages $(k_1 + \ldots + k_n = t - 1)$, then

(3)
$$P(w_0 | \eta_1, \ldots, \eta_{t-1}) = \frac{P(\eta_1, \ldots \eta_{t-1} | w_0)}{\sum_{w \in W} P(\eta_1, \ldots, \eta_t | w)} = C \cdot \prod_{i=1}^{n} P_{w_0(i)}$$

It is sufficient to prove that for every pair of frequencies

$$k_r \leqslant k_s$$

$$P(\eta_t = r | \eta_1, \ldots \eta_{t-1}) = \sum_{w \in W} \prod_{i=1} p_i^{k_{w(i)} + \delta_{w(i),r}}$$

(4)

$$P(\eta_t = \Delta | \eta_1, \ldots, \eta_{t-1}) = \sum_{w \in W} \prod_{i=1} p_i^{k_{w(i)} + \delta_{w(i),s}}$$

where $\delta_{i,j}$ is the Kronecker's symbol. Inequality (4) follows by comparing the terms on the left and right hand side, which belong to pairs (w,w^*) of permutations with the following properties:

There on $i < j \leqslant n$ are two fixed natural numbers for which

$$w(i) = r \qquad w(j) = s$$
$$w^*(i) = s \qquad w^*(j) = r$$

and $w(k) = w^*(k)$ for every $k \neq i,j$.

The second assertion of Lemma follows from the symmetry of inequality (4). Lemma 1 and the Bellman equation written in the form (2) gives.

THEOREM 1. For the uniform a priori distribution ξ of the parameter w, and for every N the optimal Bayesian sequential decision procedure is the "least frequently used" stategy.

References

[1] P. A. Franaszek and T. J. Wagner, Some distribution-free aspects of paging algorithm performance Journ. ACM. 21 (1974), 1, 31-39.

[2] P. J. Denning, Virtual memory, Computing Surveys, Vol. 2, N^{0}3, (1970), 153-89.

[3] M. Arató, On optimal performance of page storage hierarchies, to appear

[4] M.H. DeGroot, Optimal Statistical Decisions Mc Graw-Hill, (1970).

Összefoglaló

Optimális program lapolási eljárások Bayes-féle tárgyalása

Benczúr A. – Krámli A.

A dolgozat a Bayers-féle szekvenciális döntéselmélet segitségével igazolja, hogy bizonyos feltételek mellett a "legritkábban használt lap" stratégiája minimalizálja a lapolási hibák várható számát.

РЕЗЮМЕ

БАЙЕСОВСКИЙ ПОДХОД К ПРОБЛЕМЕ ОПТИМАЛЬНОГО ПОСТРАНИЧНОГО ПРОГРАММИРОВАНИЯ

А. Бенцур - А. Крамли

В работе, используя Байесовскую теорию последовательного решения, доказывается, что при некоторых условиях стратегия "наименее часто использованной страницы" минимизирует ожидаемое число страничного сбоя.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

A DETAILED, SYSTEM OVERHEAD INCLUDING, DESCRIPTION OF PRIORITY ALGORITHMS FOR REAL TIME TASK SCHEDULING

Adam Wolisz
Department of Complex Automation Systems of the Polish Academy of Sciences

1. Introdution

Author's point of view on the problem of creating an proper model describing real-time opertaing systems (O.S) has been presented in paper [1]. One can find there some characteristic features of O.S. as well as a classification of scheduling algorithms used in the real-time applications. The short overview of available theoretical results for some of these algorithms has also been provided, while a more detailed survey can be found in [2].

Up to now priority scheduling algorithms based on the minicomputers priority interrupt feature are implemented in the vast majority of O.S. Because of the hard constraints on reponse times imposed by real-time environment, a deep knowledge of their time characteristics is of essential importance, thus a proper queuing model has to be developed.

It can be easily demonstrated, that simple priority disciplinces like head of the line or preemptive resume disciplines, are not sufficient to obtain such a model. Under the head of the line discipline high-priority jobs can not receive desired response times. On the othet hand, mainly because of synchronization objectives and data integrity reasons, the use of strictily preemptive priority disciplines is not possible. During the execution of almost every programme there are some parts which should not be preempted, and which are therefore executed after disabling the priority interrupt system.

So the execution of a programme can be treated as service given in phases of either preemptive or nonreemptive type.
The existance of any given phase in the exacuted programme may also depend on data upon which the current exacution is fulfilled.

Implementation of various queuing disciplines is connected with some overhead which should be also included in the analysis. This overhead is usually caused by activities fulfilled in the privileged mode and as such can not be interrupted by any demand no matter of what priority.

The study of models reflecting the above mentioned features was started in [3] and then developed under more general assumptions in [4] and [5].
However all those investigations took place for the case when demands for programme's exacution belonged to a Poisson stream, thus open queuing systems had to be considered. This assumption is not justified, as in the majority of real-time systems one has to deal with demand's sources of finite dimension.
Corresponding to this situation closed queuing systems are difficult to analyze especially for the case of complex service disciplines, like multiphase priority service mentioned earlier.

Operational parameters of those systems, like waiting times for various demands, are often estimeted by the analysis of a corresponding system with either infinite-dimensional demand's sources (leading to an upper bound) or one dimensional demand's sources (which gives an lower bound).

The relative error caused by such an estimation in the first case has been investigated by Bazen and Goldberg [6], and according to their results may achieve even several hundred percents. On the other hand, it can be proved that in the case when one-dimensional estimation is used the relative error may not exceed one hundred percent.

Besides, one dimensional sourced describe often the actual situation existing in technological plants.

Queuing systems with one dimensional sources create also a proper model for multiprogramming systems with a fixed number of nonhomogeneus jobs, having independent I/0 facilities, as it was pointed out in [7].

Such a systems have been investigated so far only in the case of simple priority algorithms and for the processor sharing model. In this paper the analysis of a generalized priority discipline corresponding to the real-time requirements is presented for th case of N independent one-dimensional sources of demands.

2. Description of the considered model

We shall consider a single server queuing system with N one-dimensional sources generating demands $z_1, z_2, \ldots, z_k, \ldots, z_N$ respectively. The time spent in the source by the demand z_k is a stochastic variable having an exponential distribution with parameter λ_k. The demand z_k (called herafter k-type) will have priority over the demand

$$z_\lambda \quad \text{if} \quad k < I.$$

The service of demand z_k is given in phases denoted F_k^i. Any phase F_k^i may be either of preemptive type (if $i \epsilon I_k$) or of nonpreemptive type (if $i \epsilon J_k$), where I_k, J_k are proper exclusive stes.

Everly phase has a given probability of it's execution r_k^i and an arbitrary conditional probability density function(pdf) of it's execution time $s_k^i(x)$.

Further we shall use frequently the total pdf $S_k^i(x)$ given by

$$(2.1.) \qquad S_k^i(x) = \delta(x)[1 - r_k^i] + s_k^i(x) \cdot r_k^i,$$

where $\delta(x)$ is the Dirac function.

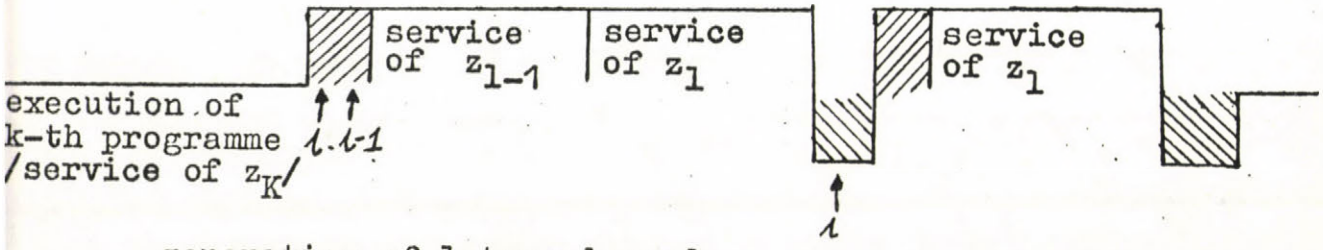
The joint pdf of k-th programme execution $S_k(x)$ can be obtained basing on S_k^i.

If for every $i \neq j$, S_k^i and S_k^j are statistically independent, then *)

$$\bar{S}_k(s) = \prod_{i \epsilon (I_k \vee I_k)} \bar{S}_k^i(s).$$

In other cases $\bar{S}_k(s)$ must be derived according to the form of dependence between S_k^i and $S_k^j (i,j \epsilon \{I_k \vee J_k\}, \quad i \neq j)$.

For the preemptive phase F_k^i, $i \epsilon I_k$ we shall assume the overhead connected with preemption and the overhead connected with resuming the preempted execution to be of nonpreemptive type and have the lenght given by arbitrary pdf's $Q_k^i(x)$ and $R_k^i(x)$ respectively. We shall also assume that if during R_k any demand of type 1, $1 < k$ will be generated, it shall be regarded as next, independent preemption as presented in fig. 1.



generation of l-type demand

Fig.1 Illustration of the assumptions according the overhead during the preemptive phases

For the nonpreemptive phases F_k^i, $i \epsilon J_k$ we shall assume, that if during the service of the phase F_k^i any demand of higher priority was generated, then after finishing F_k^i an overhead with arbitrary pdf $Q_k^i(x)$ should take place. Only after that the service of higher priority demands can be initiated.

This model will be investigated using methods similar to presented in [9] and all concepts which will not be defined here precisely are used in the meaning of [9]. In this paper the outline of the analisis and main results are given, while all additional detailes are described in [2].

* Let $f(t)$ be any given pdf. We shall denote by $f(s)$ the Laplace transform of $f(t)$

$$\bar{f}(s) = \int_0^\infty e^{-st} f(t) dt.$$

and by f any sample from the distribution $f(t)$.

Our considerations will consist of three steps. First we shall discuss the busy period for some basic schemes, where the server deals only with demands generated by a single source. Afterwards the busy period for all types of demands will be investigated.
Finally we shall describe the full process consisting of a sequence of busy and idle periods consecutively. We shall also investigate the waiting times for demands of various types.

3. Basic service schemes

Let us now discuss busy period parameters for four basic schemes of serving demands from one source with parameter λ which require service having a pdf $S(x)$.

a.) A scheme with regeneration

We shall assume, that the server works in a following mode. At time $t = 0$ a first service starts, and goes on for some time Z. A part V of this service time is devoted to an "incorrect" service in the sense, that after completion of the service, server has to be regenerated. This regeneration takes some time y depending on V in the way described be a conditional pdf $r(y/V)$. Both the times Z and V are given by a joint pdf $f_{ZV}(Z,V)$. During the regeneration a consecutive demand can be generated, which service can start only after completing of the regeneration.
Let us denote by $p_m(x,t,Z,v)dxdv\,dZ$ the pdf of the following event: at time t service is in progress with the elapsed service of the demand being equal to x, $x < x^* < x + dx$, while the service will take time Z^* including time V^* of "incorrect" service, $Z < Z^* < Z + dZ$, $v < v^* < v + dv$ $q_m(m,y,t\ v)dyd\,v$ -the joint pdf of the following events: At time t regeneration of the server is in progress for a period equal to y^*, and the "incorrect" part of the last fulfilled service took time V^*, $y < y^* < y + dy$, $V < V^* < V + dV$, $m = 0,1$ denotes the number of demands waiting for service during the regeneratin.
$p_m(t)$- the pdf of the service starting at time t
Directly from this definitions yields

$$p_m(t) = \int_0^\infty \int_0^\infty \int_0^\infty p_m(x,t,Z,v)dxdv\,dZ$$

Operation of the considered system can be described by following partial differential equations:

$$(3.1) \qquad [\frac{\partial}{\partial t} + \frac{\partial}{\partial x}]p_m(x,t,Z,V) + p_m(x,t,Z,V)\eta(x) = 0,$$

$$(3.2) \qquad [\frac{\partial}{\partial t} + \frac{\partial}{\partial y} + \eta^{\diamond}(y/V)]q_m(1,y,t,V) = \lambda q_m(0,y,t,V),$$

$$(3.3) \qquad [\frac{\partial}{\partial t} + \frac{\partial}{\partial y} + \eta^{\diamond}(y/V) + \lambda]q_m(0,y,t,V) = 0,$$

where

$$\eta(x) = \frac{f_z(x)}{1 - \int_0^x f_2(u)du} \tag{3.4}$$

and

$$f_z(x) = \int_0^\infty f_{ZV}(x,V)dV$$

$\eta(x)$ is the conditional pdf of finisching the service if the elapsed service time is equal to x Similarly

$$\eta^\diamond(y/V) = \frac{r(y/V)}{1 - \int_0^y r(u/V)du} \tag{3.5}$$

is the pdf of regeneration time completion after time y.

Equations (3.1) – (3.3) are to be solved with respect to boundary conditions:

$$p_m(0,t,Z,V) = f_{ZV}(Z,V) \int_0^\infty \int_0^\infty q_m(1,y,t,V)\eta^\diamond(y/V)dydV, \tag{3.6}$$

$$q_m(0,0,t,V) = \int_0^\infty p_m(Z,t,Z,V)dZ, \tag{3.7}$$

$$q_m(1,0,t,V) = 0, \tag{3.8}$$

and initial condition

$$p_m(x,0,Z,V) = \delta(x)f_{ZV}(Z,V). \tag{3.9}$$

We shall also define the pdf of the length of busy period, which for this case can be found as

$$b_m(t) = \int_0^\infty \int_0^\infty q_m(0,y,tV)\eta^\diamond(y/V)dydV. \tag{3.10}$$

Solving equations (3.1) – (3.3) one can obtain [2],

$$\bar{p}_m(s) = \bar{a}(s) = \frac{1}{1 - \bar{c}(s) + \bar{H}(s+\lambda)}, \tag{3.11}$$

$$\bar{p}_m(x,s,Z,V) = e^{-sx} e^{-\int_0^x \eta(u)\,du} \cdot \bar{a}(s) \cdot f_{ZV}(Z,V), \tag{3.12}$$

$$\bar{b}_n(s) = \bar{a}(s) \cdot \bar{H}(s+\lambda) = \frac{\bar{H}(s+\lambda)}{1 - \bar{c}(s) + \bar{H}(s+\lambda)}, \tag{3.12}$$

where

$$\bar{H}(s+\lambda) \doteq \int_0^\infty f_{ZV}(s,V)\bar{r}[(s+\lambda)/V]dV, \tag{3.13}$$

$$(3.14)\qquad \bar{c}(s) = \int_0^\infty \bar{f}_{ZV}(s,V)r(s,V)dV.$$

Let us now discuss the waiting time in such a service scheme. For calculating the waiting time distribution in open service schemes the following reasoning is frequently used:
Due to the Poisson stream of demands, the probability of consecutive call's generation is constant and equal to λdt (where λ is the parameter of the input stream), so one can consider with proper probabilities all possible situations occuring during the service, (eg. idle and busy periods, etc.).

Using the same methods with closed queuing systems, one can obtain so called virtual waiting time (of [8]) reflecting the period which would spend in the queue a fictitious demand, not influancing system's operation, and having the proper priority.

The real waiting time is different from the virtual waiting time, because in the case of one-dimensional sources there is no possibility of generation of the demand z_k if that demand is either in the queue or is being served.

In this paper we shall consider the real waiting time (called shortly "waiting time") exelusively.

Let us notice, that in the service scheme with regeneration the demand starting every busy period is served without waiting, while consecutive service can start only after some waiting time. The mean number of services given during one busy period is equal to

$$(3.15)\qquad \int_0^\infty p_m(t) = \bar{p}_m(0) = \frac{1}{\bar{H}(\lambda)}$$

and the mean number of services preluded by waiting in queue is equal to

$$\frac{1}{\bar{H}(\lambda)} - 1 = \frac{1 - \bar{H}(\lambda)}{\bar{H}(\lambda)}$$

The queuing phenomenon is caused in this scheme only by existance of the regeneration. We shall calculate the waiting-time distribution, assuming that the conditional pdf $r(y/V)$ is of a specific form, namely

$$(3.16)\qquad r(y/V) = r^*(y/V)[1 - \psi(V)] + \delta(y)\psi(V),$$

where $\psi(v)$ is some given function. This assumption will be discussed later. The stochastic variable V has a pdf

$$f_V(V) = \int_0^\infty f_{ZV}(Z,V)dZ = \bar{f}_{Z,V}(0,V).$$

thus if regeneration takes place, than it's length has an unconditional pdf given by (3.17).

$$(3.17)\qquad r^\diamond(y) = \int_0^\infty r^*(y/V) \cdot f_V(V)dV.$$

The waiting time can be in this case determined as the result of substraction of two stochastic variables, describing the lenght of the regeneration and the generation process respectively, under the condition that a demand will be generated during the regeneration. So the pdf of waiting time is equal to

$$\omega_m^{-I}(\Theta) = \lambda \frac{\bar{r}^{\diamond}(\Theta) - \bar{r}^{\diamond}(\lambda)}{\lambda - \Theta} \cdot \frac{1}{1 - \bar{r}^{\diamond}(\lambda)} \tag{3.18}$$

where $[1 - \bar{r}^{\diamond}(\lambda)]$ is the probability of demand's generation during the regeneration of a non-zero lenght.
As functions of the shape (3.18) will be often used in further considerations, we shall introduce

$$\overline{F_{\lambda}}(\varphi,\Theta) = \lambda \frac{\bar{\varphi}(\Theta) - \bar{\varphi}(\lambda)}{\lambda - \Theta} \cdot \frac{1}{1 - \bar{\varphi}(\lambda)}$$

describing the pdf of waiting time caused by some process having the pdf $\varphi(t)$ if the demands are generated according to the negative exponential distribution with parameter λ.
So

$$\bar{\omega}_m^{I}(\Theta) = \bar{F}_{\lambda}(\bar{r}^{\diamond}, \Theta).$$

Finally the total pdf of waiting time in the service scheme with regeneration is given by (3.19)

$$\begin{aligned}\bar{\omega}_m(\Theta) &= \frac{\dfrac{1 - \bar{H}(\lambda)}{\bar{H}(\lambda)}}{\dfrac{1}{\bar{H}(\lambda)}} \cdot \lambda \frac{\bar{r}^{\diamond}(\Theta) - \bar{r}^{\diamond}(\lambda)}{\lambda - \Theta} \frac{1}{1 - \bar{r}^{\diamond}(\lambda)} + \frac{1}{\dfrac{1}{\bar{H}(\lambda)}} \cdot 1 = \\ &= [1 - \bar{H}(\lambda)] \cdot \lambda \frac{\bar{r}^{\diamond}(\Theta) - \bar{r}^{\diamond}(\lambda)}{(\lambda - \Theta)[1 - \bar{r}^{\diamond}(\lambda)]} + \bar{H}(\lambda) \cdot 1 = \\ &= [1 - \bar{H}(\lambda)] \cdot \bar{F}_{\lambda}(r^{\diamond}, \Theta) + \bar{H}(\lambda) \cdot 1.\end{aligned} \tag{3.19}$$

b.) A scheme with initial process and regeneration
In this scheme at $t = 0$ an initial period having a pdf $\Omega(x)$ starts. If during the Ω no demand is generated the busy period is supposed to complete (a short busy period). On the other hand if during Ω the demand is generated, then after completing the initial period it's service starts, (a long busy period).
As, similarly to the case a.) the service includes some "incorrect"part and causes perhaps a necessity of regeneration, the process may continue as in the previous scheme.

The busy period distribution $b_m^{\Omega}(t)$ can be found using the results from previous scheme,

$$b_m^{\Omega}(t) = e^{-\lambda t} \cdot \Omega(t) + (1 - e^{-\lambda t}) \cdot \Omega(t) * b_m(t),$$

and after transformation

$$(3.20)\quad \overline{b}_m^{\Omega}(s) = \overline{\Omega}(s+\lambda) + [\overline{\Omega}(s) - \overline{\Omega}(s+\lambda)] \cdot \overline{b}_m(s).$$

Similarly one can obtain $p_m^{\Omega}(t)$ (being the analogon of $p_m(t)$), observing, that

$$(3.21)\quad p_m^{\Omega}(t) = (1 - e^{-\lambda t}) \cdot \Omega(t) \cdot p_m(t),$$

which after transformation and utilizing (3.11) yields

$$(3.22)\quad \overline{p}_m^{\Omega}(s) = \frac{\overline{\Omega}(s) - \overline{\Omega}(s+\lambda)}{1 - \overline{c}(s) + \overline{H}(s+\lambda)} = \frac{\overline{\Omega}(s) - \overline{b}_m^{\Omega}(s)}{1 - \overline{c}(s)}$$

$\overline{H}(s+\lambda)$ and $\overline{c}(s)$ used in (3.22) are given by (4.13) and (4.14).

The mean number of services given during a busy period can be found utilizing $p_m^{\Omega}(t)$, as being equal to

$$(3.23)\quad \int_0^{\infty} p_m^{\Omega}(t) = \overline{p}_m^{\Omega}(0) = \frac{1 - \overline{\Omega}(\lambda)}{H}.$$

In the considered scheme every service is preluded by waiting in queue, however queuing before the first service has a different lenght than the consecutive ones. Speaking more precisely, $[1 - \overline{\Omega}(\lambda)]$ services (that means the first one if there is any service at all) come after the waiting time determined by the initial period, thus having the pdf equal to $F_{\lambda}(\Omega,\Theta)$. All remaining services can be fulfilled after waiting time identical as in scheme a.), having the pdf equal to $F_{\lambda}(r^{\diamond}, \Theta)$.

The pdf of total waiting time can be for this scheme experessed by (3.)

$$(3.24)\quad \overline{\omega}_m^{\Omega}(\Theta) = \frac{1 - \overline{\Omega}(\lambda)}{\overline{H}(\lambda)} \overline{F}_{\lambda}(\Omega,0) + \frac{[1 - \overline{H}(\lambda)][1 - \overline{\Omega}(\lambda)]}{\overline{H}(\lambda) \cdot [1 - \overline{\Omega}(\lambda)]} \cdot \overline{F}_{\lambda}(r^{\diamond}, \Theta) =$$

$$= \overline{H}(\lambda) \cdot \overline{F}_{\lambda}(\Omega,\Theta) + [1 - \overline{H}(\lambda)] \cdot \overline{F}_{\lambda}(\overline{r}^{\diamond}, \Theta).$$

Let us notice, that schemes a.) and b.) discussed here are a generalization of simple schemes given in [1].
Results presented there can be obtained from formulas obtained above, by following subtitutions:

$$\overline{r}(s/V) = 1 \qquad \text{or} \qquad f_{ZV}(Z,V) = f_Z(Z) \cdot \delta(V).$$

c.) A scheme with set-up time and regeneration.

In this scheme we shall assume, that first service in each busy period must be preceded by a set-up time having the pdf $\Omega(y)$ begining after the generation of the demand starting that busy period.

Every service can cause the regeneration, exatcly as it was in the previous schames.

It can easily proved that $b_m^{\Omega,1}$ and $p_m^{\Omega,1}$ (describing for that scheme the lenght of busy period, and the begining of a consecutive service are to derived from equations (3.25) and (3.26),

$$\overline{b}_m^{\Omega,1}(s) = \overline{\Omega}(s) = \overline{\Omega}(s) \cdot \overline{b}_m(s) \tag{3.26}$$

$$\overline{p}_m^{\Omega,1}(s) = \overline{\Omega}(s) \cdot \overline{p}_m(s). \tag{3.27}$$

where $\overline{p}_m(s)$ and $\overline{b}_m(s)$ are given by (3.11) and (3.12).

The mean number of services given during a single busy period is equal to $\frac{1}{\overline{H}(\lambda)}$.

In this scheme the first service in every busy period is preluded by a waiting time equal to Ω, while all consective services are given after waiting time having the pdf equal to $\overline{F}_\lambda(r^\diamond,\Theta)$.

Of course $r^\diamond$ is given as previously by (3.17).

Thus, the total pdf of waiting time can be calculated from (3.27) .

$$\overline{\omega}_m^{\omega,1}(\Theta) = \frac{1}{\frac{1}{\overline{H}(1)}} \cdot \overline{\Omega}(\Theta) + \frac{\frac{1-\overline{H}(\lambda)}{\overline{H}(\lambda)}}{\frac{1}{\overline{H}(\lambda)}} \cdot \overline{F}_\lambda(r^\diamond,\Theta) = \overline{H}(\lambda) \cdot \overline{\Omega}(\Theta) + [1-\overline{H}(\lambda)] \cdot \overline{F}_\lambda(\overline{r}^\diamond,\Theta) \tag{3.27}$$

d.) A scheme with initial period, set–up time and regeneration

In this scheme at $t = 0$ the initial period having a pdf $\Omega(y)$ starts and if during this period a demand is generated, the set-up time with pdf $\Omega(x)$ begins after completion of the initial period, and then the service takes place (a long busy period). If during the initial period no demand is generated, busy period will be finished (a short busy period). Service paremeters similar to the pervious ones can be obtained easily also for this scheme:

$$b_m^{\Omega,\Omega}(t) = e^{-\lambda t}\Omega(t) + (1 - e^{-\lambda t})\Omega(t) * b_m(t) * \Omega(t), \tag{3.28}$$

$$p_m^{\Omega,\Omega}(t) = (1 - e^{-\lambda t}) \cdot \Omega(t) * p_m(t) * \Omega(t), \tag{3.29}$$

where $b_m(t)$ and $p_m(t)$ are given by (3.12) and (3.11).

After transformation we obtain (utilizing (3.22)

$$\overline{b}_m^{\Omega,\Omega}(s) = \overline{\Omega}(s+\lambda) + [\overline{\Omega}(s) - \overline{\Omega}(s+\lambda)] \cdot \overline{b}_m(s) \cdot \overline{\Omega}(s), \tag{3.30}$$

and

$$(3.31)\qquad \overline{p}_m^{\Omega,v}(s) = \overline{\Omega}(s) \cdot \frac{\overline{\Omega}(s) - \overline{\Omega}(s+\lambda)}{1 - \overline{c}(s) + \overline{H}(s+\lambda)} \quad .$$

The mean number of services given during a single busy period is equal to

$$\int_0^\infty p_m^{\Omega,v}(t) = \frac{1 - \overline{\Omega}(\lambda)}{\overline{H}(\lambda)} \quad ,$$

including $[1 - \overline{\Omega}(\lambda)]$ services preluded by the waiting time with pdf equal to $\overline{\Omega}(\Theta) \cdot \overline{F}_\lambda(\Omega,\Theta)$

and $\dfrac{[1 - \overline{H}(\lambda)][1 - \overline{\Omega}(\lambda)]}{\overline{H}(\lambda)}$ services preluded by waiting time with pdf equal to

$\overline{F}_\lambda(r^\diamond, \Theta)$. The justification is for this case identical as in the earlier schemes. Finally, the pdf of total waiting time is given by (3.32)

$$(3.32)\qquad \overline{w}_m^{\Omega,v}(\Theta) = \overline{H}(\lambda) \cdot \overline{\Omega}(\Theta) \cdot \overline{F}_\lambda(\Omega,\Theta) + [1 - \overline{H}(\lambda)] \cdot \overline{F}_\lambda(r^\diamond, \Theta)$$

In furter considerations we shall utilize the results obtained for the basic service schemes in the analysis of our priority system. It will be therefore necessary to specify the obtained formulas for a given type of demands, which will be denoted by an additional subscript.

For example $b_{m,3}^{\Omega}$ will denote the busy period in the scheme b.) after substituting λ_3, S_3, c_3, H. Of course for calculating H_3 and c_3 from formulas (3.1) and (3.1) one has to determine previously the functions $f_{ZV,3}(Z,V)$ and $r_3(y/V)$.

4. The joint busy period end server's utilization factor

After presenting the results describing basic service schemes we shall return to the model introduced in section 2.

We shall define completion time c_k^i of the phase F_k^i as time period between the beginning of execution of F_k^i until the servoce station becoms ready to start service of the same type demand's next phase. Quite similarly can be defined the completion time of the full demand c_k.

As it is well known in the priority queues theory, introducing of the completion time makes it possible to regard the existance of higher priority demands.

We shall assume, for the time being, the full knowledge of pdfs for c_K^i, $i \epsilon (I_K \vee J_K)$, $K = 1,2, \ldots, N$.

The detailed analysis of c_K^i will be given later in this section. For simplification of further analysis we shall assume, that the service of demand z_K consists of a finite number of phases, equal to k_K, and that the last phase is of nonpreemptive type,*) i.e. $k_K \epsilon J_K$.

Let γ_K denote the joint busy period for demands $z_1, z_2, \ldots, z_K$ γ_K can start in two possible ways:

- either because of the generation of z_K, which service starts immidiately, being however influanced by the higher priority demands. All the preemptive phases F_K^i, $i \epsilon I_K$ may be preemptived, while the nonpreemtive phases F_K^i, $i \epsilon J_K$ may be followed by the service of higher priority demands. Thus, if the last phase in the service of z_K is nonpreemptive, the demand z_K can be generated once more during the service of higher-priority demands following this phase.
- or because of the generation of any one of demands $z_1, z_2, \ldots, z_{K-1}$ starting the busy period γ_{k-1}, which in turn creates initial period for the process described above.

According to these remarques γ_K can be derived through an application of the basic schemes a.) and b.) in which the completion times of all but last one of the demand's z_K service phases form jointly the "correct" part of the service, while tha last, nonpreemptive phases $F_K^{k_K}$ is considered as the "incorrect" service V.

This incorrect service may couse the necessity of regeneration, i.e. the necessity of serving some hihger priority demands generated during $F_K^{k_K}$, where according to the earlier introduced denotation, k_K is the number of the last, nonpremptive phase. Following this reasoning we can formulate:

$$(4.1) \qquad \overline{\gamma}_K(s) = \frac{\lambda_K}{\Lambda_K} \cdot \overline{b}_{m,K}(s) + \frac{\Lambda_{K-1}}{\Lambda_K} \cdot \overline{b}_m^{\gamma_{K-1}}(s),$$

where $\Lambda_K = \sum_{i=1}^{K} \lambda_i$.

For obtaing $b_{m,K}^{\gamma_K - 1}$ it is necessary to determine $f_{ZV,K}$ and $r_K(y/V)$ which in this case are of the folowing form:

$$(4.2) \qquad f_{ZV,K}(Z,V) = C_K^*(Z) * \delta(Z,-V) \cdot S_K^{k_K}(V),$$

* Let us notice that if the last phase would not be of nonpreemptive type then we can always add one artifical phase with r_K^i and $s_K^i(x) = \delta(x)$, i being the number of the additional phase.

$$(4.3)\qquad \overline{r}_K(s/V) = e^{-\Lambda_{K-1}V} \cdot \delta(y) + \frac{1 - e^{-\Lambda_{K-1}V}}{1 - \Omega^*(\lambda_{K-1})}\, e^{sV} \cdot$$

$$\cdot\, [\overline{\gamma}_{K-1}^{\Omega^*}(s) - \overline{\Omega}^*(s + \Lambda_{K-1})],$$

where

$$(4.4)\qquad \overline{c}_K^*(s) = \prod_{i=1}^{k_{K-1}} \overline{c}_K^{\,i}(s),$$

$$(4.5)\qquad \overline{\Omega}^*(s) = e^{-sV} \cdot \overline{Q}_K^{\,k_K}(s).$$

We can observe that $\overline{r}_K(s/V)$ is of the form suggested in (3.16) where

$$\psi_K(V) = e^{-\Lambda_{K-1}V}$$

and

$$(4.6)\qquad \overline{r}_K^*(s/V) = \frac{1}{1 - \overline{\Omega}^*(\Lambda_{K-1})}\, e^{sV}[\overline{\gamma}_{K-1}^{\Omega^*}(s) - \overline{\Omega}^*(s + \Lambda_{K-1})].$$

We shall now consider completion times for phases of both types a.) completion time for preemptive phases

During the service time S_K^i of the phase F_K^i there can occure n preemptive with probability π_n

$$\pi_n = \frac{(\Lambda_{K-1} \cdot y)^n}{n!}\, e^{-\Lambda_{K-1}y}$$

We shall assume that every preemption lasts for time x, with pdf equal tp $\Gamma_{K-1}(x)$ Therefore

$$(4.7)\qquad c_k^i(x) = \int_0^x \sum_{n=0}^{\infty} e^{-\Lambda_{k-1}y}\, \frac{(\Lambda_{k-1}y)^{n^*}}{n!}\, \Gamma_{k-1}^n(x - y) \cdot S_k^i(y)dy,$$

where n^* denotes n time convolution.
After transformation

$$(4.8)\qquad \overline{c}_K^{\,i}(s) = \overline{S}_K^i[s + \Lambda_{K-1} \cdot E(\Gamma_{K-1}^i)],$$

We shall now derive $\Gamma_{K-1}(s)$ basing on assumptions acording the overhaed time.

From Fig 1 one see that Γ_{K-1} can be divided into two parts:

– the first one, denotd by g_K^i from generation of the demand, until the overhead R begins for the first time, and

– the second one, denoted by σ_K^i from the instant when R begins for the first time, until it is finished succesfully (that mean without any demand of higher priority being generated during R).

As the two parts are independent, we can present $\overline{\Gamma}_{K-1}(s)$ in the following way

(4.9) $$\Gamma_{K-1}^i(s) = \overline{g}_K^i(s) \cdot \overline{\sigma}_K^i(s).$$

The first factor is equal to a following sum

(4.10) $$\underset{K}{g^i}(s) = \sum_{j=1}^{K-1} \frac{\lambda_j}{\Lambda_{K-1}} \overline{\Phi}_{K-1}^j(s)$$

where

$$\overline{\Phi}_l^j(s) = \begin{cases} \overline{b}_{m,l}^{\overline{\Phi}_{l-1}^j(s)}(s) & \\ & d/a \quad j < l \leqslant k-1 \\ \overline{b}_{m,j}(s) \cdot \overline{\gamma}_{j-1}^{Q_K^i}(s) & d/a \quad l = j \end{cases}$$

(4.11) $$\overline{\gamma}_l^{\Omega}(s) = \overline{b}_{m,l}^{\gamma_{l-1}^{\Omega}}(s)$$

Formula (4.10) reflects the order of service, in the case when F_K^i was interrupted by the demand z_j, which occures with probability $\frac{\lambda_j}{\Lambda_{K-1}}$
In that case the overhead Q takes place, during which some demands of priority higher than j can be generated, and only after completing Q the service of demands can start in the sequence determined by their priorities.
On the other hand $\overline{\sigma}_K(s)$ can be derived from equation (4.12)

(4.12) $$\overline{\sigma}_K^i(s) = \overline{R}_K^i(s + \Lambda_{K-1}) + [1 - \overline{R}_K^i(\Lambda_{K-1})] \cdot \frac{\overline{\gamma}_{K-1}^{\Omega^*}(s) - \overline{\Omega}^*(s + \Lambda_{K-1})}{1 - \overline{\Omega}(\Lambda_{K-1})} \cdot \sigma_K^i(s)$$

where

$$\overline{\Omega}^*(s) = \overline{R}_K^i(s) \cdot \overline{Q}_K^i(s)$$

Formula (4.12) describes the two possible situations:
either no demand of higher priority than K is generated during R_K^i (this case is given by the first component) or at least one of them is generated, starting a quite complicated process. According to the assumptions about the overhead, R_K^i is in this case followed by Q_K^i, and both these times of overhead create the initial period for the busy period γ_{K-1}, which is a "long" one.

After completing γ^{Ω}_{K-1} (which can be derived from the iterative formula (4.11)) the overhead R^i_K starts again, begining ones more the whole process.

b.) completion time for nonpreemptive phase

For obtaining the completion time c^i_K, $i \in J_K$ we can use the same reasoning as in the case of σ^i_K, thus obtaining

$$(3.13)\qquad \overline{c}^{\,i}_K(s) = \overline{S}^i_K(s + \Lambda_{K-1}) + [1 - \overline{S}^i_K(\Lambda_{K-1})]\,\frac{\overline{\gamma}^{\Omega}_{K-1}(s) - \overline{\Omega}(s + \Lambda_{K-1})}{1 - \overline{\Omega}(\Lambda_{K-1})}\,,$$
$$i \in J_K$$

Now we are in the position to derive $f_{ZV,K}(Z,V)$ and $r_K(y/V)$ according to formulas (4.2) and (4.3), because all c^i_K for $i \in (I_K V J_K)$, $K = 1,2,\ldots,N$ can be calculated from either (4.8) or (4.13).

We can also obtain $\overline{c}_K(s)$ directly from (3.14) as well as γ_K from (4.1).

So far we have considered the busy periods, exclusively. Th general service process consists of a sequence of busy periods and idle periods and the busy periods starting instant can be treated as regeneration points of a renewal process.

Assuming that at $t = 0$ a busy period starts, the transform of renewal density function $\overline{h}_K(s)$ can be calculated as

$$(4.14)\qquad \overline{h}_K(s) = \frac{\Lambda_K \overline{\gamma}_{K(s)}}{s + \Lambda_K[1 - \overline{\gamma}_K(s)]}$$

Let $e_K(t)$ be the probability of system being idle at time t of the general process, with respect to demands $z_1, z_2, \ldots, z_K$.

Using the renewal argument it is easy to obtain

$$(4.15)\qquad \overline{\hat{e}}_K(s) = \frac{1}{s + \Lambda_K[1 - \overline{\gamma}_K(s)]}\,,$$

Let us notice that $[1 - \hat{e}_K(t)]$ determines the ulilization factor of the server.

For $t \to \infty$ the stationary state probability of the server being idle is

$$(4.16)\qquad \hat{e}_K = \frac{1}{1 + \Lambda_K E(\gamma_K)}\,.$$

Basing on (4.16) one can easily find the fraction of server's time spend on dealing with demands z_K $(K = 1,2,\ldots,N)$, that means needed for serving this demand and for the overhead connected with it's preemption and resuming of the service.

This factor is equal to $\hat{e}_{K-1} - \hat{e}_K$ and is influanced by all the demand's parameters and also by the service discipline.

5. Waiting time distributions

The discussion of waiting time distribution will be presented for the stationary state, and propen formulas will be obtained by a detailed analysis of service processes occuring the busy period.

For estimating $W_K^{(N)}(\tau)$ the pdf of waiting time of the demand z_K in a system serving demands $z_1, z_2, \ldots, z_K, \ldots z_N$, we shall first investigate the number of services n_K given to demand z_K during a single busy period γ_N.
We shall also consider waiting times preluding respective services. $n_K^{(N)}$ will be derived using an iterative formula

$$k = 1,2, \ldots, N$$

$$(5.1) \qquad n_K^{(k)} = \frac{\Lambda_{k-1}}{\Lambda_k} \cdot n_k^{(k-1)} + M_k N_K^k$$

$$K = 1,2, \ldots, k$$

where M_k denotes the number of demands z_k served during a busy period γ_K, and can be found equation (5.2)

$$(5.2) \qquad n_k^{(k)} = M_k = \frac{\lambda_k}{\Lambda_k} \cdot \overline{p}_{m,k}(0) + \frac{\Lambda_{k-1}}{\Lambda_k} \overline{p}_{m,k}^{\gamma_{k-1}}(0),$$

after a reasoning similar to those which justified (4.1).

Using $N - K$ times (5.1) we obtain

$$(5.3) \qquad n_K^{(N)} = \frac{\Lambda_K}{\Lambda_N} n_K^{(k)} + \sum_{L=K+1}^{N} \frac{\Lambda_L}{\Lambda_N} M_L N_K^L$$

Both in (5.1) and (5.3) N_K^L denotes a mean number of demands z_K served during the completion time c_K, and has to be calculated with respect to the multiphase structure of demand's z_L service.
Thus

$$(54) \qquad N_K^L = \sum_{i \in (I_L \vee J_L)} r_L^i \cdot N_K^{L,i}$$

where $N_K^{L,L}$ denotes a mean number of demands z_K served during the completion time c_L^i.
Let us notice, that generation of demand z_K is possible only if that demand is neither served nor waiting in the queue.
Thus the generation of K type demand during c_L^i can happen in the instant when

- the service of phase F_L^i is in progress
- overhead Q_L^i or R_L^i is in progress
- the busy period γ_{K-1} is in progress
- the completion time of an type 1 demand ($l = K + 1, K + 2, \ldots L - 1$) is in progress

Denoting by $m_K^{L,i}$ the number of services given to demands z_K generated in the first three of obove listed situations leads to the following formula describing

$$N_K^{L,i} = m_K^{L,i} + \sum_{l=K+1}^{L-1} m_l^{L,i} \; N_K^l \tag{5.5}$$

We shall now given the formulas for calculating $m_K^{L,i}$ seperately for preemptive and non-preemptive phases, providing them only with short comments, while the full eduction is given in [2].

a.) preemptive phases

For F_L^j, $j \epsilon L_L$ we obtain

$$\begin{aligned} m_K^{L,j} = \frac{1}{r_L^j} \nu_L^j \{ & \lambda_K \bar{p}_{m,K}(0) + \Lambda_{k-1} \bar{p}_{m,K}^{A}(0) + \\ & + (\Lambda_{L-1} - \Lambda_K) \bar{p}_{m,K}^{\gamma_{K-1}^{Q_L^j}}(0) \} + \frac{\nu_L^j}{r_L^j} \frac{\Lambda_{L-1}}{\bar{R}_L^j(\Lambda_{L-1})} \{ [1 - \bar{R}_L^j(\Lambda_{K-1})] \bar{p}_{m,K}^{\Omega^{\mathrm{I}}}(0) \\ & + [\bar{R}_L^j(\Lambda_{K-1}) - \bar{R}_L^j(\Lambda_{L-1} - \lambda_K)] \bar{p}_{m,K}^{\Omega^{\mathrm{II}}}(0) + \\ & + \bar{R}_L^j(\Lambda_{K-1} - \lambda_K) \bar{p}_{m,K}^{\Omega^{\mathrm{III}}}(0) \} \end{aligned} \tag{5.6}$$

where

$$A = \sum_{i=1}^{K-1} \frac{\lambda_i}{\Lambda_{K-1}} \cdot \bar{\Phi}_{K-1}^i(s), \qquad \nu_L^j = E(S_L^j)$$

$$\bar{\Phi}_l^i(s) = \begin{cases} \bar{b}_{m,l}^{\varphi_L^i - 1}(s) & \text{for } i < l \leqslant K-1 \\ \bar{b}_{m,i}(s) \cdot \bar{\gamma}_{i-1}(s) & \text{for } l = i \end{cases}$$

$$\bar{\Omega}^{\mathrm{I}}(s) = \frac{\bar{\gamma}_{K-1}^{\Omega^*}(s) - \bar{\Omega}^*(s + \Lambda_{K-1})}{[1 - \bar{\Omega}^*(\Lambda_{K-1})]} \quad \text{for } \bar{\Omega}^*(s) = \bar{R}_L^j(s) \cdot \bar{Q}_L^j(s)$$

$$\bar{\Omega}^{\mathrm{II}}(s) = \frac{\bar{R}_L^j(s + \Lambda_{K-1}) - \bar{R}_L^j(\Lambda_{L-1} - \lambda_K + s)}{\bar{R}_L^j(\Lambda_{K-1}) - \bar{R}_L^j(\Lambda_{L-1} - \lambda_K)} \cdot \bar{\gamma}_{K-1}(s)$$

$$\bar{\Omega}^{\mathrm{III}}(s) = \frac{\bar{R}_L^j(\Lambda_{L-1} - \lambda_K + s)}{\bar{R}_L^j(\Lambda_{L-1} - \lambda_K)}$$

The right side of (5.6) consitss of six components liked in two groups. In the first group there are considered the cases when z_K is generated during

- the service of F_L^j
- the busy period caused by generation of the demands $z_1, z_2, \ldots, z_K$
- the overhead Q_L^j caused by the generation of demands $z_{K+1}, z_{K+2}, \ldots, z_{L-1}$ or during the service of any demand having the priority higher than K which could be generated during Q_L^j.

The second group contains the possibilities of z_K generation during following processes, connected with tha part σ_L^j of preemption (cf section 4), namely the possibilities that:

- during R_L^j at least one demand of priority higher than K is generated, causing the overhead Q_L^j and a proper busy period
- during R_L^j no demand of priority higher than K, and at least one demand of priority between K and L is generated.
- during R_L^j no demand of priority different than K is generated.

For evaluating the number of services given to z_K in any of this situations the basic schemes from section 3 where used. In fact, for the first situation the basic scheme c.) is a proper one for the last situation we have to utilize the scheme d.), while all other can be described applying the basic scheme b.) with due substitutions.

Discussing the basic schemes we have pointed out, the way of obtaining mean number of services given in every scheme, as well as possible waiting time distributions.
Using the derived in section 3 formulas with substitutions adapting them to all listed above situations we can observe, that the waiting time can have one of the seven possible density functions, given in (5.7).

$$
(5.7)\quad \begin{cases}
\overline{U}^{L,j}_{K,1}(\Theta) = \overline{\gamma}^{Q^j_L}_{K-1}(\Theta) \\
\overline{U}^{L,j}_{K,2}(\Theta) = \overline{F}_{\lambda_K}(A,\Theta) \\
\overline{U}^{L,j}_{K,3}(\Theta) = \overline{F}_{\lambda_K}(\gamma^{Q^j_L}_{K-1},\Theta) \\
\overline{U}^{L,j}_{K,4}(\Theta) = \overline{F}_{\lambda_K}(\Omega^{I},\Theta) \\
\overline{U}^{L,j}_{K,5}(\Theta) = \overline{F}_{\lambda_K}(\Omega^{II},\Theta) \\
\overline{U}^{L,j}_{K,6}(\Theta) = \overline{\gamma}^{Q^j_L}_{K-1}(\Theta)\cdot \overline{F}_{\lambda_K}(\Omega^{III},\Theta) \\
\overline{U}^{L,j}_{K,7}(\Theta) = \overline{F}_{\lambda_K}(r^{\diamond},\Theta)
\end{cases}
$$

Let us notice that Ω^{I}, Ω^{II} and Ω^{III} where defined in (5.6) while $r^{\diamond}$ was given ba (3.17) and (4.6).

Functions $U^{L,j}_{K,i}$, $j\epsilon I_L$, $i\epsilon[1,2,\ldots,6]$ describe the waiting times connected with the first service in everyone of the six considered situations, and $U^{L,j}_{K,7}$ gives the waiting time preluding the services caused by regeneration

Let us denote the number of services preluded with waiting time having the equal to $U^{L,j}_{K,i}$ by $[m^{L,j}_K]_i$.

Introducing adequate substitutions to the formulas given in section 3 it is easy to show, that

$$
(5.8)\quad
\begin{cases}
[m_K^{L,j}]_1 = \dfrac{\nu_L^j}{r_L^j}\,\lambda_K \\[2ex]
[m_K^{L,j}]_2 = \dfrac{\nu_L^j}{r_L^j}\,\Lambda_{K-1}[1-\bar{A}(\lambda_K)] \\[2ex]
[m_K^{L,j}]_3 = \dfrac{\nu_L^j}{r_L^j}\,(\Lambda_{L-1}-\Lambda_K)[1-\bar{\gamma}_{K-1}^{\varphi_L^j}(\lambda_K)] \\[2ex]
[m_K^{L,j}]_4 = \dfrac{\Lambda_{L-1}\nu_L^j}{r_L^j\cdot\bar{R}_L^j(\Lambda_{L-1})}\,[1-\bar{R}_L^j(\Lambda_{K-1})][1-\bar{\Omega}^{\mathrm{I}}(\lambda_K)] \\[2ex]
[m_K^{L,j}]_5 = \dfrac{\Lambda_{L-1}\nu_L^j}{r_L^j\,\bar{R}_L^j(\Lambda_{L-1})}\,\{[\bar{R}_L^j(\Lambda_{K-1})]-[\bar{R}_L^j(\Lambda_{K-1})]\}[1-\bar{\Omega}^{\mathrm{II}}(\lambda_K)] \\[2ex]
[m_K^{L,j}]_6 = \dfrac{\Lambda_{L-1}\nu_L^j}{r_L^j\,\bar{R}_L^j(\Lambda_{L-1})}\,\bar{R}_L(\Lambda_{K-1}-\lambda_K)[1-\bar{\Omega}^{\mathrm{III}}(\lambda_K)] \\[2ex]
[m_K^{L,j}]_7 = \dfrac{\nu_L^j}{r_L^j}\,\dfrac{1-\bar{H}_K(\lambda_K)}{\bar{H}_K(\lambda_K)}\,\{\lambda_K+\Lambda_{K-1}[1-\bar{A}(\lambda_K)]+ \\
\quad +(\Lambda_{L-1}-\Lambda_K)[1-\bar{\gamma}_{K-1}^{\varphi_L^j}(\lambda_K)]\}+ \\
\quad +\dfrac{\Lambda_{L-1}\cdot\nu_L^j}{r_L^j\cdot\bar{R}_L^j(\Lambda_{L-1})}\cdot\dfrac{1-\bar{H}_K(\lambda_K)}{\bar{H}_K(\lambda_K)}\,\{[1-\bar{R}_L^j(\Lambda_{K-1})][1-\bar{\Omega}^{\mathrm{I}}(\lambda_K)]+ \\
\quad +[\bar{R}_L^j(\Lambda_{K-1})-\bar{R}_L^j(\Lambda_{L-1}-\lambda_K)][1-\bar{\Omega}^{\mathrm{II}}(\lambda_K)]+ \\
\quad +\bar{R}_L^j(\Lambda_{K-1}-\lambda_K)[1-\bar{\Omega}^{\mathrm{III}}(\lambda_K)]\}
\end{cases}
$$

Obviously

$$m_K^{L,j} = \sum_{i=1}^{7} [m_K^{L,j}]_i \qquad j \epsilon I_L$$

b.) nonpremptive phases

The reasoning for the case of nonpreemptive phases F_K^j, $j \epsilon J_L$ is quite similar to that applied to the previous case, in the part concerning resuming the preempted service, i. e. the part σ_K^j of the preemption. Thus introducing S_k^j instead of R_L^j in the last three situations investigated during creation of (5.6) we obtain

$$(5.9) \qquad m_K^{L,j} = [1 - s_L^{-j}(\Lambda_{K-1})]\overline{p}_{m,k}^{\Omega\,IV}(0) + [\overline{s}_L^{\,j}(\Lambda_{K-1}) - \overline{s}_L^{\,j}(\Lambda_{L-1} - \lambda_K)]\overline{p}_{m,K}^{\Omega\,IV}(0) + + \overline{s}_L^{\,j}(\Lambda_{K-1} - \lambda_k)\,\overline{p}_{m,K}^{\Omega\,III}(0).$$

where

$$\overline{\Omega}^{IV}(s) = \frac{\overline{\gamma}_{K-1}^{\Omega^*}(s) - \overline{\Omega}^*(s + \Lambda_{K-1})}{1 - \overline{\Omega}^*(\Lambda_{K-1})} \qquad \text{for } \overline{\Omega}^*(s) = \overline{s}_L^{\,j}(s) \cdot \overline{\varphi}_L^j(s)$$

$$\overline{\Omega}^{V}(s) = \frac{\overline{s}_L^{\,j}(s + \Lambda_{K-1}) - \overline{s}_L^{\,j}(s + \Lambda_{L-1} - \lambda_K)}{\overline{s}_L^{\,j}(\Lambda_{K-1}) - \overline{s}_L^{\,j}(\Lambda_{L-1} - \lambda_K)}\ \overline{\gamma}_{K-1}^{\varphi_j^L}(s)$$

$$\overline{\Omega}^{VI}(s) = \frac{\overline{s}_L^{\,j}(\Lambda_{L-1} - \lambda_K + s)}{\overline{s}_L^{\,j}(\Lambda_{L-1} - \lambda_K)}$$

In the similar way we can list also the pdf describing all possible waiting times, $U_{K,i}^{L,j}$ and mean numbers of services preluded by the waiting time with respective

pdf $-\ [m_k^{L,j}]_i \quad i\epsilon\{1,2,3,4\}, \quad j\epsilon J_k, \quad K = 1,2,\ldots,N.$

$$(5.10) \quad \begin{cases} \overline{U}_{K,1}^{L,j}(\Theta) = \overline{F}_{\lambda_K}(\Omega^{IV},\Theta) \\ \overline{U}_{K,2}^{L,j}(\Theta) = \overline{F}_{\lambda_K}(\Omega^{V},\Theta) \\ \overline{U}_{K,3}^{L,j}(\Theta) = \overline{F}_{\lambda_K}(\Omega^{VI}\Theta) \cdot \gamma_{K-1}^{\Theta_L^j}(\Theta) \\ \overline{U}_{K,4}^{L,j}(\Theta) = \overline{F}_{\lambda_K}(r_k^{\diamond},\Theta) \end{cases}$$

$$(5.11)\quad \begin{cases} [m_K^{L,j}]_1 = [1 - \bar{s}_i^{\,j} \cdot (\Lambda_{k-1})][1 - \bar{\Omega}^{IV}(\lambda_K)] \\ [m_K^{L,j}]_2 = [\bar{s}_L^{\,j}(\Lambda_{K-1}) - \bar{s}_L^{\,j}(\Lambda_{L-1} - \lambda_k)][1 - \bar{\Omega}^{V}(\lambda_K)] \\ [m_K^{L,j}]_3 = \bar{s}_L^{\,j}(\Lambda_{K-1} - \lambda_K)[1 - \bar{\Omega}^{VI}(\lambda_k)] \\ [m_K^{L,j}]_4 = \dfrac{1 - \bar{H}_K(\lambda_K)}{\bar{H}_K(\lambda_K)} \{[m_K^{L,j}]_1 + [m_K^{L,j}]_2 + [m_K^{L,j}]_2 + [m_K^{L,j}]_3\} \end{cases}$$

Surely enough

$$m_K^{L,j} = \sum_{i=1}^{4} [m_K^{L,j}]_i \qquad j \epsilon J_L$$

After obtaining formulas 5.6. and 5.9 we are in position to calculate n_K^N – the mean number of demands z_K served during a busy period γ_N.

We have also discussed the waiting times preluding services of demand z_K given during the completion time of demand z_L, $L = K + 1, K + 2, \ldots, N - 1$. For obtaining the total pdf of waiting time we have to investigate additionally waiting times preluding the service of z_K during the busy period γ_K, that means preluding the $n_K^{(K)}$ services during in (5.2). One can observe that the considered process can be described either by the basic scheme a.) with probability $\dfrac{\tilde{\lambda}_K}{\Lambda_K}$ or by the basic scheme b.) with probability $\dfrac{\Lambda_{K-1}}{\Lambda_K}$.
Using formulas derived for those basic scheme it is easy to find that the waiting times may have following pdfs:

$\bar{U}_{K,1}^{(K)}(\Theta) = 1$ – if the demand z_K started the busy period γ_K

$\bar{U}_{K,2}^{(K)}(\Theta) = \bar{F}_{\lambda_K}(\gamma_{K-1}, \Theta)$ – if γ_K has been started by the demand of priority higher than K, and demand z_K is generated during the so initiated γ_K

$\bar{U}_{K,3}^{(K)}(\Theta) = \bar{F}_{\lambda_K}(r_K^{\diamond}, \Theta)$ – in the case of second and further generations of demand z_K during the busy period γ_K (regenerations)

Denoting by $[n_K^{(K)}]_i$ the number of services preluded by waiting time with pdf equal to $U_{K,i}^{(K)}$ we obtain

$$[n_K^{(K)}]_1 = \frac{\lambda_K}{\Lambda_K}$$

$$[n_K^{(K)}]_2 = \frac{\Lambda_{K-1}}{\Lambda_k}\,[1 - \bar{\gamma}_{K-1}(\lambda_K)]$$

$$[n_K^{(K)}]_3 = \frac{1 - H_K(\lambda_K)}{\bar{H}_K(\lambda_K)} \cdot \frac{\lambda_K}{\Lambda_K} + \frac{\Lambda_{K-1}}{\Lambda_K}[1 - \bar{\gamma}_{K-1}(\lambda_K)]$$

Obviously

$$\eta_K^{(K)} = \sum_{i=1}^{3} [\eta_K^{(K)}]_i$$

Finally we can derive the full formula for waiting time distribution considering with respective probabilities all discussed so for possible waiting time distributions.
Thus the pdf of waiting time $W_K^{(N)}(\tau)$ has a Laplace transform equal to

$$(5.13) \qquad \overline{W}_K^{(N)}(\Theta) = \frac{1}{\eta_K^{(N)}} \frac{\Lambda_K}{\Lambda_N} \cdot \sum_{j=1}^{3} [\eta_K^{(K)}]_j \cdot U_{K,j}^K + \sum_{L=K+1}^{N} \frac{\Lambda_L}{\Lambda_N} M_L \cdot \omega_K^L$$

$$\omega_K^L = \sum_{i \epsilon (I_L \vee J_L)} r_L^i \cdot \omega_K^{L,i}$$

$$\omega_K^{L,i} = \nu_K^{L,i} + \sum_{l=K+1}^{L-1} \eta_L^{L,i}\, \omega_K^l$$

$$\nu_K^{L,i} = \sum_{j=1}^{7} [m_K^{L,i}]_j \cdot U_{K,j}^{L,i} \qquad d/a \quad i \epsilon I_l$$

$$\nu_K^{L,i} = \sum_{j=1}^{4} [m_K^{L,i}]_j \cdot U_{K,j}^{L,i} \qquad d/a \quad i \epsilon J_L$$

6. Final remaremarks

In this paper the analysis of a complex priority queuing system with one dimensional sources of demands has been presented. Using the reported results it is possible to investigate a wide range of priority algorithms as the considered model includes as special cases not only such simple disciplines as preepmtive resume and head of the line discipline (with or without overhead).

Basing on this model one can also determine easily theservice parameters of the discretionary priority discipline, as well as investigate priority systems where service is given in quanta os a predetermined value. Such priority systems with time slicing (and properly chosen quanta for every priority level) make it often possible to quarantee desired response times for demands of various types with overhead smaller than obtained in the case of other disciplines.

The special cases of the considered model can be obtained by proper defining of

$$r_K^i,\ S_K^i,\ I_K \text{ and } J_K \text{ for } K = 1,2,\ldots,N.$$

The comparison of various scheduling disciplines for the case of one dimensional sources, basing on the considered model will be presented in a seperate paper, as will as the losses of computer's throughput due to overhead.

Finally let us notice that the analisy was done under quite general assupmtions, allowing arbitrary distributions of service times, and overhead. The results were obtained in the form of Laplace transforms of probability distributions and not only in terms of mean values.

References

[1] A. Wolisz, "Real Time Operating Systems Probabilistic Models" Proceedings of the First Winter School on Mathematical Problems of Operating Systems, Közlemények 15/1975, 55-71.

[2] "Foundations of selection and analysis of plant-oriented operating systems", Vol. I. Single processor plant-oriented operating systems. A collective study. Research report, Department of Complex Automation systems, Polisch Academy of Sciences, Gliwice, December 1975 (in Polish).

[3] L. Schrage "A mixed – priority queue with applications to the analysis of real – time systems" Operations Research, Vol 17 No 4 pp. 728-741.

[4] A. Wolisz , "Modeling real–time operating systems as a single server multiphase queuing systems "Podstawy Sterowania Vol VI., No 2 (in Polish)

[5] A. Wolisz "A single server priority queuing model of real – time operating systems," Preprints of the International Conference "Informatica 75", paper No 1.14

[6] J.P.Buzen, P.S. Goldberg "Guidelines for the use of infinite source queuing models in the analysis of computer sytem performance", SJCC 1974, AFIPS Vol 43 pp 771-774.

[7] J. Tomkó "Some queueing models in the mathematical destription of computer systems" Közlemények 15/1975 pp 55-71

[8] N.K. Jaiswal "Priority Queues", Academic Press New York 1968.

Összefoglaló

"Real time" prioritásos rendszerek vizsgálata

Adam Wolisz

A dolgozat a prioritásos rendszer valószinüség elméleti vizsgálatával foglalkozik.

Р Е З Ю М Е

Пробный анализ приоритетных алгоритмов диспетчеризации задач при работе в реальном масштабе времени.

Воллиш А.

В статье предлагается некоторая одноканальная система приоритетного обслуживания с постоянными приоритетами как модель операционной системы вычислительной машины.

В этой модели обслуживание любой заявки происходит в фазах, которые могут подвергаться дисциплине с абсолютным или относительным приоритетом.

Предлагается, что заявки на обслуживание генерированные одномерными источниками заявок.

Исследуется время потраченное на прерывание текущей программы,а также на возврат в прерванную программу.

Принимая, что любую фазу описывает вероятность ее выполнения и любое расположение времени выполнениея, получаем параметры очередей в случае пуассоновских потоков заявок.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

A DYNAMIC ADAPTIVE CONTROL FOR BATCH PROCESSING AND TIME–SCHARING MODES

Pal Tőke
Computer and Automation Institute of Hungarian Academy od Sciences

I. Introduction

A characteristic feature of the third generation is the apparition of simultaneity, i. e. potential or real existence of logically independent activities.

It is the consequence of simultaneity, further of sharing and multiplexing that the events do not occour according to deterministic conformity to the rules. We can take for instance the random process of demands for the use of a given resource, e.g. the central unit, moreover the temporal, random behaviour of the extent of claims that arise. That model can be generalized: In general a resource is given, it can be software type or hardware type alike, demands for use arive at random and the extent of the demand cannot be described in a deterministic way. Demands can be met in an order depending on the character of the resources. The most frequent modes are sharing and multiplexing (fictive multiplication), but there are also resources for which the queueing methods can be investigated with classical models of queueing theory.

The arising demands, if the resource has no free capacity, make a queue or queues on a predetermined or dynamically fitted basis. The construction of queues or the handling of the already constructed queues is supported by special hardware instructions.

The paper deals with questions of modelling operating systems by means of queueing theory. It shows how investigations made on theoretical models can be used for the design of operation systems supporting various modes of application or for the "tuning" of an already existing system.

II. Description of processes by stochastic methods

As mentioned in the introduction, methods of queueing and network theory play a substantial role in modelling the run of processes.

Processes are virtually successions in time of demands for resources. As a resource it is understood here a software – or hardware – type resource in the most general sence. e.g. the use of external store, memory, CPU, compilers, use of special operating system capability etc.

The microstructure of the process can be characterized by the following random vector

$$[R_1\tau_1, R_2\tau_2, \ldots, R_N\tau_N]$$

where

$R_i \epsilon R$ (set of resources).

τ_i variables with time dimension of random character. It gives the occupation time of the resource (the size of the demand).

N the quantity of stochastic character typical for the given process.

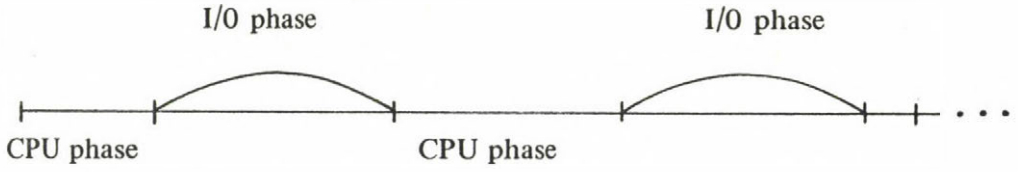
It is a fundamental task to give the process vectors for the functional modelling of operating systems.

For the sake of analytical handling it is generally assumed that the transition from to has Markov-type properties. Furthermore it is often assumed that quantities are exponentially distributed variables.

Modelling can be achieved in complete generality with the help of the methods of network theory.

By limiting ourselves to the simplified model we distinguish essentially two kinds of resource types: the CPU and the I/0 devices. Accordingly, the demand can be CPU demand and I/0 demand. This simplified model lends itself particularly well to describe computer processes under very much extended conditions.

The running in time of the process is illustrated by the following figure:



The simultaneous existence of processes, independent in a logical sence, results in their simultaneous arrival in both CPU phase and I/0 phase becoming possible, so that investigations relating to questions of the run of processes are virtually restricted to examinations of queueing systems.

The following are required to make a given queueing model:

1.) The stating of the input process.
2.) The stating of the priority discipline.
3.) The stating of the size of the entering demand.
4.) The description of the serving mechanism.

The novel serving mechanisms achieved by the operating systems have raised particular problems, not investigated before the appearance of operating systems, for the queueing theory. These will be dealt later on more in detail.

III. Characterization of the priority disciplines of batch processing and time-scharing modes

The serving mechanism is determines by the feature of the resource and the mode of utilization, while the priority discipline is decisively determined by the mode of utilization.

In case of batch processing a minimal running time has to be assigned to the given batch programs. This purpose has to be substantially supported by the priority strategy.

In case of time-sharing modes the situation is quite different. Intelligent terminal lines are connected with the central computer and on the particular lines the maintenance of the communication at a prefixed speed level is absolutely necessary. By the speed characteristic for given lines we understand the speed ratio compared with other lines. The primary task of the priority discipline is to secure the communication speed rations. In the practice it is uniformity that kas to be secured most often. It has to be noted that whether the batch processing or the time-sharing modes are considered, in themselves they are non-existent idealized borderline cases.

Let us take a practival example. In the Computer and Automation Institute of Hungarian Academy of Sciences the CDC 3300 computer operates for most of the time batch processing mode, its telecommunication lines are of remote batch terminal type. At the same time we can circumscribe with the central computer many processes for which the maintenance of a given uniform speed is absolutely necessary from the point of view of the balanced operation of the whole operating system. Such are e.g. the input preprocessings and the output post-processings, console communication and other operating system tasks.

It is very important that the priority disciplines contain the possibility of meeting these essential demands. More will be said in this respect below when the concrete realization will be discussed.

IV. Methods of practical realization of priority systems

From the investigation of concrete cases we can generally infer that the priority level of the processes is in general of static character and depends explicitely on the statement of the external parameter (or parameters).

Let us consider e. g. the MASTER operating system. The user himself defines the priority levels of the tasks pertaining to the job by stating the CLASS parameter. (This seemingly does not contradict the caracter of the static statement as the running priority is in general the sum of the priorities of the calling and called tasks). The four basic priority levels are the C, I, S, B levels, resp. the E class. When stating the priority level the characters of the program tasks ought to be primarily considered. But in case of the MASTER the priority level is characteristic for the job and is given among the parameters determining the job. At the same time a job can consist of several program tasks of various nature whose priority levels had to be determined just by the character of the given task. There is, however, no possibility for this. May I note here that the priority levels determined by the characters of the programs tend to

support the uniform run of program tasks of various nature. Here "uniform" means the nearly identical speed. It can occur that this tendency does not agree with the priority discipline characteristic of the users' mode and its effect is showing just there where it can be disadvantageous.

The more recent MASTER versions contain a new feature, the dynamic load balance. Its essence is the following: To each successive CPU and I/0 phase belongs a priority index proportional to the CPU size. The priority index is determined by the system after the I/0 phase, immediately before the beginning of the CPU phase. The running priority occurs before the direct beginning of the serving of the CPU demand in such a way that the running priority for the following CPU and I/0 phasis is the average of the immediately preceding four priority indices.

Let us examine critically the above algorithm. As it can be seen, the running priority is virtually determined by the CPU demands so that to the short CPU demands a great running priority belongs. The setving mechanism of the central unit works, however, also in a strict sence according to a Round Robin algorithm where priority has no essential part. The running priority therefore plays an essential role only in the construction of the series of I/0 demands. Yet this circumstance can also contradict the priority discipline characteristic of the given mode of processing.

V. **Dynamic adaptive control**

Practical experiences show that the primary purpose of the users is not specify exactly, according to the nature of the given task the priority class, on the contrary, they are moved by their individual interest to obtain as great running priority for their task as possible. There is plenty of theoretical reasonning and case studies to be read in the international technical literature about distortions due to high priority, especially about high priority paralysis as a result of unaccountable use of the emergency class. Such an obvious phenomenon is the memory thrashing problem. For the emergency class is not concerned by the thrashing limit of the memory, the memory is oversaturated by active programs without an adequate quantity of resources, in this case memory, being engaged for the particular programs.

An answer to the questions raised above has to be found in the dynamic adaptive mode defining the implicit priority. As our ideas have to be realizad within the frame of an already existing structure, the theoretical models quotee as motivations are from a certain pount of view predetermined and the realization of the algorithm to be constructed shoud be adjusted to an already existing structure.

Starting from the fact that a batch processing mode has to be virtually achieved it has also to be considered that certain processes are to be provided with uniform speed.

The priority strategy characteristic of the batch processing mode should in the first place achieve the parallel operation with maximum intensity of the elements suitable for parallel operation. This requires first an adequate starting strategy and priority system supported

implicitly by parameters characteristic of the job. Two basic parallel controls exist for the configuration investigated: main control for the CPU and block control to co-ordinate the I/0 system. The block control controls the I/0 system parallel with the multiplexing method.

A starting strategy in conformity with the above criteria should therefore meet the requirement of the peripheral devices. As according to the given structure the start takes place on a priority basis, we preceed as follows: The class of the jobs using the slow peripherals should be B, that of those using the fast peripherals I, the priority level of jobs not using the peripherals should be C. On account of other considerations the jobs using peripherals of mixed type are at a disadvatage.

The priority strategy has to be such as to secure a uniform distribution for the number of demands in the queuing systems concerned. This can be approximated if we strive for a minimal number of demands to be at disposal in the particular systems. This means in the practice that shorter demands should get higher priority.

It is abvious that such a priority strategy depends on the momentaneous state of the system, therefore the models are analytically difficult to be handled. The optimal priority strategy is the Shortest Remaining Processing Time First Strategy (SRPT) the demonstration of which can be found in the paper of L. Schrage [1] but it follows also of other theoretical considerations.

The SRPT discipline cannot be realized in every queueing system. Realizability greatly depends on the character of the resource. Difficulty arises also in cases where the demand of service is a priori known.

Under very general assumptions the following relation holds:

$$L = \lambda \cdot w$$

where

L is the average number of demands in the system,
w is the average value of the time of stay in the system,
λ is the intensity of arrival.

The above relation was first proved by J.D.C. Little [3] under very general assumptions, both enlightening and rather likely to be verified in practice, too if we wish to rely in the course of the demonstration on the methods of renewal theory. Figuratively speaking we assume that the initial load period lasts for a finite time, the moment of the evacuation of the system is a recurrent event, further that the system, beginning with the moment of the evacuation, "forgets its past". The exact conditions can be found e. g. in the paper of W.S. Jewell [4].

In cases where the extent of the demand is a priori known the situation is somewhat simpler. Such is e. g. the I/0 system. When an I/0 demand arises, the size of the block to be moved is given and the knowledge of the speed of peripherals enables us to estimate the dimension of the demand.

The situation is quite different in the case of the CPU. The arising demand is not a priori known. Other technical devices are needed, viz. an appropriate serving mechanism. The first serving mechanism which enabled faster runs to be achieved for shorter CPU demands in comparison with longer CPU demands, was the Round Robin service.

In order to critically evaluate the RR algorithm we need first analytical results. By now it can be said that the one-channel RR service mechanism systems are due to analytical methods completely settled both in case of a process with finite source capacity and in case of an infinite source capacity (Poisson input).

The problem was dealt with by many people, among others [7, 8, 9, 10, 11] papers. The first and at the same time most generalized was given by Lajos Takács in his paper [5, 6]. Unfortunately this paper is very rarely referred to.

The essence of the method is that it succeeded to make to the model an analog one to the same from a certain point of view, classic at the same time. It succeeded to find the condition of the stationary state, further exact formulas for the stationary distribution of element in the system, finally the stationary distribution for the running time of the demand. L. Takács assumed a Poisson arrival but its method is applicable in case of input of finite source capacity (model of finite number of processes existing simultaneously).

The principal results are the following:

Theorem 1. (*L. Takács*): *If* $\lambda\alpha < q$ *then the process* $\{\xi(t),\ 0 \leqslant t < \infty\}$ *has a unique stationary distribution* $P(\xi(t) = j) = P_j^{*}(j = 0,1, \ldots)$ *and for* $|z| \leqslant 1$

$$U^{*}(z) = \sum_{j=0}^{\infty} P_j z^j = (1 - \frac{a\alpha}{q}) \frac{q(1-z)\Psi(\lambda(1-z))}{(q + pz)\psi(\lambda(1-z)) - z}$$

where

$$\psi(s) = \int_0^{\infty} e^{-sx} dH(x)$$

P *is the probability of the return*
$\xi(t)$ *the lenght of queue at time* t
$H(x)$ *distribution function of the quantum size*

The distribution of $\xi(t)$ queue size for the process of type $[F(x), H(x), p]$ is the same as for the process

$$[F(x), H^{*}(x), 0)$$

where

$$H^{*}(x) = q \sum_{k=1}^{\infty} p^{k-1} H_k(x)$$

$H_k(x)$ denotes the k^{th} iterated convolution with itself.

If χ_n denotes the time needed to complete the current service at the instant $t = \tau_k - 0$ (immediately before the arrival of the customer), ξ_n denotes the queue size at the same moment, then $\{\xi_n, \chi_n;\ n = 1,2, \ldots\}$ is a Markov sequence.

Using the following symbols for stationary distribution

$$P_j = P(\xi_n = j) \qquad (j = 0,1,2, \ldots)$$

$$P_j(x) = P(\chi_n \leqslant x,\ \xi_n = j) \qquad (j = 1,2,3, \ldots)$$

$$\Pi_j(s) = \int_0^\infty e^{-sx}\, dP_j(x) \qquad (j = 1,2,3, \ldots)$$

Theorem 2. (*L. Takács*): *If* $\lambda\alpha < q$, *then the Markov sequence* $\{\xi_n, \chi_n;\ n = 1,2, \ldots\}$ *has a unique stationary distribution for which*

$$U(s,z) = \sum_{j=1}^{\infty} \Pi_j(s) z^j = \left(1 - \frac{\alpha\lambda}{q}\right) \frac{\lambda z(1-z)[\Psi(s) - \Psi(\lambda(1-z))]}{[z - (q + pz)\Psi(\lambda(1-z))][s - \lambda(1-z)]}$$

Θ_n *denotes the total time spent in the system by* n^{th} *customer.*

Theorem 3. (*L. Takács*): *If* $\lambda\alpha < q$, *then* Θ_n *has a unique stationary distribution* $P(\Theta_n \leqslant x)$, *which is given by the following L. S. transform*

$$\phi(s) = q \sum_{k=1}^{\infty} p^{k-1} U_k(s,1) \qquad (Re(s) \geqslant 0)$$

where

$$U_1(s,z) = P_0 \Psi(s + \lambda(1-z)) + U(s + \lambda(1-z), (q + pz)\Psi(s + \lambda(1-z)))$$

for

$$Re(s) \geqslant 0 \text{ and } |z| \leqslant 1,\ P_0 = 1 - \lambda\alpha/q,$$

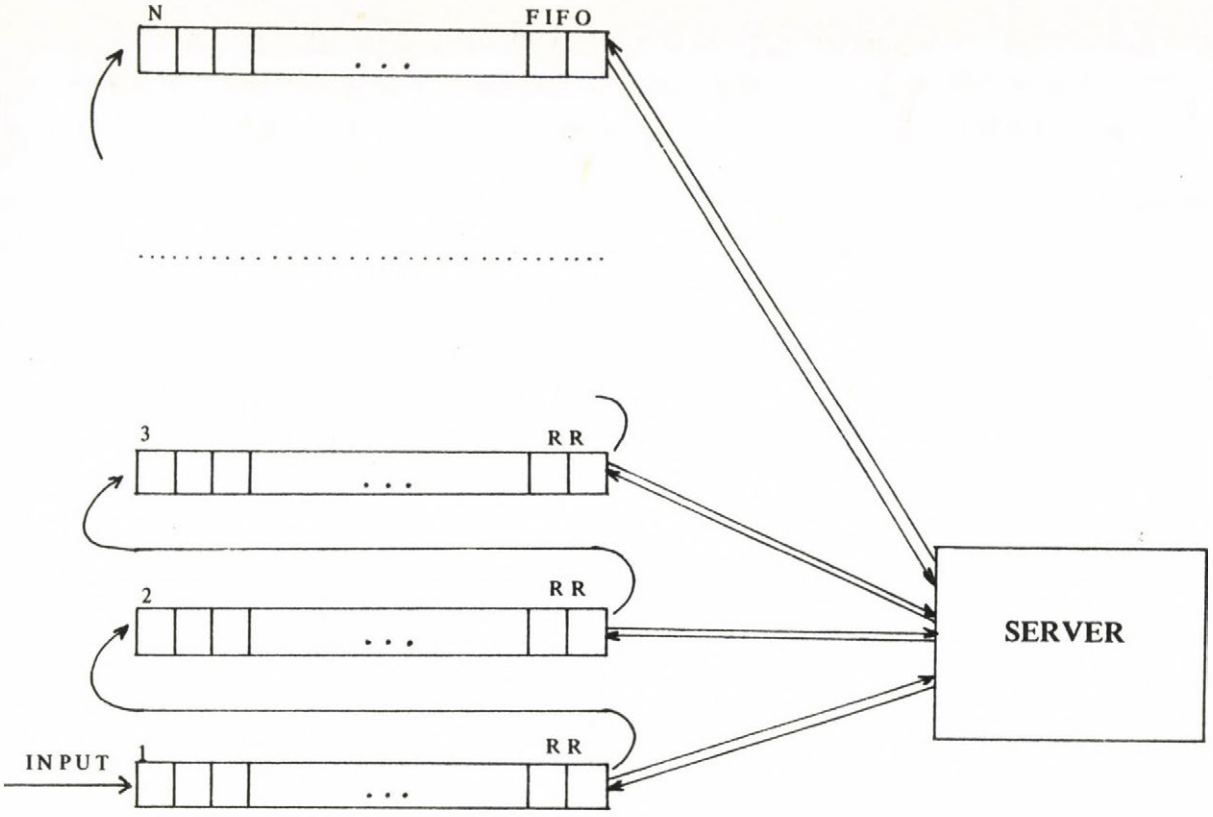
$U(s,z)$ *is defined before and*

$$U_{k+1}(s,z) = \Psi(s + \lambda(1-z)) U_k(s_1 (q + pz)\Psi(s + \lambda(1-z))) \quad \text{for} \quad k = 1,2, \ldots$$

The other papers too apply virtually the method of Takács.

The purely RR design is not appropriate for us because at the beginning every demand starts from an identical priority level and there is no possibility to grant preferential treatment to exceptional processes. Furthermore it is graphically clear that the preference granted to short CPU demands substantially decreases beyond a certain saturation of the system. Also this is undesirable.

This latter difficulty is eliminated by the so-called FB_N model.



The essence of this is that we prefer N series. After a certain quantity of service the demand ascends to a row a level higher. There is served at the particular levels in RR mode, at the highest level the service will be interrupted on a basis of continuation if a new demand arrives, else the service follows the FIFO algorithm.

The system can be analytically handled. The analytical results can be found in the paper of L. Schrage [2] in the most general way. Here the condition of the stationary state as well as the description of the stochastic behaviour of the running time can be found.

A weak point of the FB_N model is that every process starts from an identical priority level and there is no chance to grant durable preferences relative to each other to preferred processes.

In the practice such a modified variety of the FB_N can be applied in wich the entry at every level is permitted, further it is possible for any demand never to be let to ascend beyond a given level.

The FB_N^v model seems to be specially purposeful in the case of a CPU configuration where there is no chance to make efficient self-correcting algorithms because of the long time needed for self-administration. In case of the CDC 3300 computer the choice is motivated by the fact that a large store of devices for string manipulation, supported by both hardware and software, are at our disposal.

No analytical handling of the FB_N^v model can be expected. Under certain restrictions we may, however, rely on analytical results, too, which serve us as useful guidelines for the handling of preferred processes.

Concerning the I/0 system the problem is of quite another nature. The dimension of the demand is a priori known. Here we have, however, to cope with a difficulty of another kind viz. the finite number of priority levels. It follows therefrom that the task is the determination of the finite series of the priority levels.

Let us suppose that we know the distribution $F(x)$ of the arriving demands. Let the number of the priority classes be K. The question is how we can give the $0 < \sigma_1 < \sigma_2 < \ldots < \sigma_{K-1} < \infty$ numbers so that

$$\Xi(\sigma_1 \sigma_2, \ldots, \sigma_{K-1}) = \sum_{j=1}^{k} ajE(W_j^k)$$

should be the minimum.

If the arrival is Poisson process the problem can be handled also analytically and the solution provides valuable informations. In case of the arrival having finite source capacity (which would be of more importance for us) analytical results are unfortunately not known as yet.

If we wish to give even speeds to some prefered processes, we can avail ourselves of the method of balance.

The essence of the idea is the following : If in a queueing system the delay is greater than the required level, then we try to improve the situation by starting the service of the demand in the other, following queueing system from a higher priority level. In an inverse case we compensate by the use of a lower priority levele.

By delay index we understand the quotient of the dimension of the demand and the time of the of the stay in the system. In the following queueing system we decide on the basis of the delay index on the priority assignment. There are no analytical methods at our disposal which would enable us to make one to one correspondence between delay index and priority levels.
For "tuning", we can rely in the first place on empirical observation and simulation methods.

References

[1] L. Schrage, A proof of the optimality of the shortest remaining processing time discipline Operations Research, Vol. 16. (1968) 687-690 pp.

[2] L. Schrage, The queue M/G/1 with feedback to lower priority queues Management Science, Vol. 13. No. 7. March, (1967) 466-474 pp.

[3] J. D. C. Little, A prof fort the queueing formula: Operations Research, Vol. 9 (1961), 383-387 pp.

[4] W.S. Jewell, A Simple proof of Operations Research, Vol. 15. (1967) 1109-1116 pp.

[5] L., Takács, A single-server queue with feedback The Bell System Technical Journal, March, (1963) 505-519 pp.

[6] L. Takács, Priority queues Operations Research, Vol. 12. (1964), 63-74 pp.

[7] E. G. Coffman and L. Kleinrock, Feedback Oueueing Models for Time-Shared Systems Journal of ACM, Vol. 15. 1968. No. 4. 549-576. pp.

[8] L. Kleinrock, Analisys of Time Shared Processor Nav. Res. Logistics Quest. Voll 11. 1964. 59-73 pp.

[9] I. Adiri, A dynamical time-scharing priority queue Journal of ACM, Vol. 18. No. 4. 603-610 pp.

[10] I. Adiri, M. Hofri and M. Yadin, A Multiprogramming queue Journal of ACM, Vol.20. (1973). No. 589-603 pp.

[11] I. Adiri and Avi-Itzhak, A Time-Sharing Queue with a Finite Numbers of Customers Journal of ACM, Vol. 16. (1969) No. 2. 315-323 pp.

[12] M. Arató, E. Knuth, P. Tőke, On stochastic control of multiprogrammed computer systems Proc. of IFAC Conference on Stochastic Control, Budapest, 1974. 305-311 pp.

Összefoglaló

Kötegelt és kollektiv felhasználását támogató dinamikusan adaptiv vezérlés

Pál Tőke

A dolgozat a kötegelt és kollektiv felhasználást támogató üzemmódok prioritási stratégiák jellemzésével, és elvi kérdéseinek vizsgálatával foglalkozik.

Megvizsgálja a dinamikus vezérlés lehetőségét az erőforrás jellegétől függően, és részletesen taglalja a központi egység és az I/0 rendszer dinamikus vezérlésének kérdéseit. Külső paraméterektől explicit nem függő priorotási rendszert dolgoz ki. A dolgozat végezetül foglalkozik az operációs rendszerek matematikai modellezhetőségének problémáival, majd konkrét környezetben foglalkozik a kidolgozott algoritmusok realizálhatósági kérdéseivel.

Р Е З Ю М Е

ДИНАМИЧЕСКИ АДАПТИРОВАННОЕ УПРАВЛЕНИЕ РЕЖИМОМ ПАКЕТНОЙ ОБРАБОТКИ И КОЛЛЕКТИВНОГО ПОЛЬЗОВАНИЯ

Пал Тёке

В статье анализируется приоритет стратегии режимов, поддерживающих пакетную обработку и коллективное пользование, а также рассматриваются принципиальные вопросы их управления.

Исследуются возможности динамического управления в зависимости от характера ресурсов, в частности, вопросы динамического управления ЦПУ, и системой ввода-вывода.

Предлагается приоритетная система, не зависящая явно от внешних параметров задач.

В заключение рассматривается проблемы математического моделирования О. С., и приводятся результаты реализации предлагаемых алгоритмов для О.С. MASTER.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

ON THE USE OF DIFFUSION APPROXIMATIONS FOR THE CYCLIC QUEUE MODEL

Mária Rét
BHG Telecommunication Works, Budapest

Consider the following model of a multiprogrammed computer system (see Figure 1.).

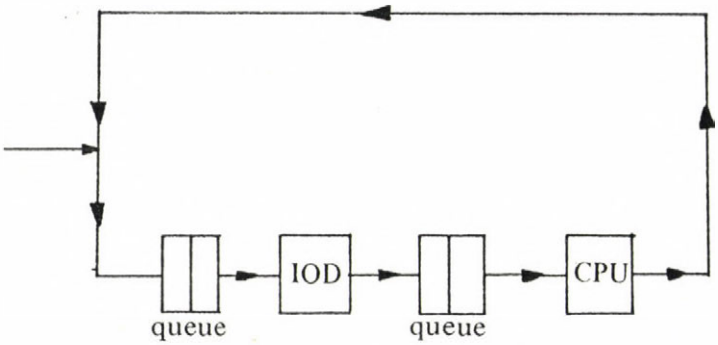


Figure 1.

This is a closed two-server system. At any instant exactly K programs are in the cycle formed by the central processor unit (CPU) and the input-output devices (IOD). Each program is either at the so-called IOD stage or at the CPU-stage (awaiting or under service). This model has been analized by many authors (see [3]-[6]) using a diffusion approximation for the number of programs in the CPU-stage.

In this note we shall give an exact proof for the validity of diffusion approximation in the case of exponentially distributed service times. We shall use the following

Notations:

$N(t)$ – the number of programs at the CPU-stage at time t,

ξ_i – the service time of the i-th demand on the IOD,

η_j – the service time of the j-th demand on the CPU,

K – degree of multiprogramming.

We assume that the following conditions are fulfilled:

1.) ξ_i and η_j, $i = 1,2, \ldots$, $j = 1,2, \ldots$ are mutually independent random variables.

2.) ξ_i, $i = 1,2, \ldots$ are independent identically distributed (i.i.d.) random variables with distribution function $P\{\xi_i \leqslant t\} = 1 - e^{-\lambda t}$.

3.) η_j, $j = 1,2, \ldots$ are i.i.d. random variables with distribution function $P\{\eta_j \leqslant t\} = 1 - e^{-\mu t}$.

Let $\Delta = 1/K$ and $X_\Delta(t) = N(t)\Delta$. Obviously $X_\Delta(t)$ is a time-homogeneous Markov process

with countable state-space. The states of $X_\Delta(t)$ are the values $x_i = i\Delta$, $i = 0,1, \ldots K$. Let us denote the interval $[0,1]$ of the real axis and the σ-algebra of their Borel-sets with $\{ \chi, \mathcal{B} \}$.

Let $\mathcal{D}_{[0,T]}(\chi)$ be the space of functions with discontinuities of the first kind, defined on the time interval $[0,T]$ and with range in the separable, complete, metrical space χ.

Let $\mathcal{L}$ be the space of real, continuous and $\mathcal{B}$-measurable functions on $[0,1]$. Let the norm on $\mathcal{L}$ be $\| f \| = \sup_x |f(x)|$, if $f \epsilon \mathcal{L}$.

We will allow Δ to tend to zero and both λ and μ to depend on Δ. Therefore we shall denote them by λ_Δ and μ_Δ, respectivelly. The following statement will be proved.

Theorem

Let $$|\lambda_\Delta - \mu_\Delta| \Delta \to m$$
$$|\lambda_\Delta + \mu_\Delta| \Delta^2 \to \sigma^2 > 0$$

if $\Delta \to 0$,

then for arbitrary constant α

$$P\{F(X_\Delta(t)) < \alpha\} \to P\{F(X_0(t)) < \alpha\},$$

F being continuous functional on $\mathcal{D}_{[0,1]}(\chi)$,

$X_0(t)$ – *is a diffusion process on the real axis with reflecting barriers at points* 0 *and* 1, *with the infinitesimal generator* $A_0 f = \sigma^2 / 2 f'' + mf'$. *The domain* $\mathcal{L}'$ *of* A_0 *is the space of twice continuously differentiable functions for which* $f'(0) = f'(1) = f''(0) = f''(1) = 0$.

Proof. The proof of this theorem is based on a Gikhman-Skorohod's theorem ([2], Vol. I, page 508). This theorem states, that $P\{F(X_\Delta(t)) < \alpha\} \to P\{F(X_0(t)) < \alpha\}$ as $\Delta \to 0$ if

1.) the finite dimensional distributions (f.d.d.) of $X_\Delta(t)$ tend to corresponding f.d.d. of $X_0(t)$ and

2.) for arbitrary $\epsilon > 0$

$$\lim_{h \downarrow 0} \overline{\lim_{\Delta \to 0}} \sup_{s - t} \{ P_\Delta(t,x,s,V_\epsilon(x)); \; x \epsilon \chi, \; 0 \leqslant s - t \leqslant h \} = 0.$$

Here $P_\Delta(t,x,s,\Gamma)$, $\Gamma \epsilon \mathcal{B}$ is the probability of transition for the process $X_\Delta(t)$ and $V_\epsilon(x) = \{ y : |y - x| > \epsilon \}$.

As $X_\Delta(t)$ is a time-homogeneous process, it is enough to prove that

$$\lim_{h \downarrow 0} \overline{\lim_{\Delta \to 0}} \{ P_\Delta(t,x,V_\epsilon(x)); x \epsilon \chi, \; 0 \leqslant t \leqslant h \} = 0.$$

Furthermore, we known that as soon as $\epsilon > \Delta$,

$$P_\Delta(t,x,V_\epsilon(x)) = o(t); \quad x \epsilon \chi, \quad 0 \leqslant t \leqslant h.$$

Thus, we can see that the second condition of Gikhman-Skorohod' theorem is satisfied. The fulfilment of the first condition is proved in the next lemma.

Lemma. *Under the conditions of the theorem the f.d.d.'s of* $X_\Delta(t)$ *tend to the f.d.d.'s of* $X_0(t)$.

Proof. Denote $T_\Delta(t)f(x)$ the transition operator (see [2] Vol. II., page 30.) corresponding to the Markov process $X_\Delta(t)$ and $T_0(t)f(x)$ the same for the process $X_0(t)$, where $f \epsilon \mathcal{L}$. To prove the lemma it is well enough to verify that

$$T_\Delta(t)f(x) \to T_0(t)f(x) \qquad \text{as} \qquad \Delta \to 0,$$

because in our case the convergence of these transition operators implies the convergence of the transition probabablities

$$P_\Delta(t,x\,\Gamma) \qquad \to \qquad P_0(t,x,\Gamma), \qquad \Gamma \epsilon \mathcal{B}$$

(where $P_0(t,x,\Gamma)$ is the transition probability of the process $X_0(t)$) and as it is known the finite dimensional distributions of the processes can be described by the help of the transition probabilities and the initial distribution.

We will use the next lemma, proved by Feller ([1], page 460.).

Approximation lemma *Let* $\{T_\Delta(t)\}$ *be a family of pseudo–Poissonian semigroups commuting with each other and generated by the endomorphisms* A_Δ. If $A_\Delta f \to A_0 f$ *for all* f *of a set* $\mathcal{L}'$ *dense in* $\mathcal{L}$, *then*

$$T_\Delta(t) \to T_0(t) \qquad \text{as} \quad \Delta \to 0,$$

where $\{T_0(t)\}$ *is a semi-group of contractions whose infinitesimal generator agrees with* A_0 *for all* $f \epsilon \mathcal{L}'$. $\mathcal{L}'$ *is the domain of the operator* A_0.

Continue the function $f(x)$ outside the interval $[0,1]$ so that for $\Delta > 0$ $f(-\Delta) = f(0)$ and $f(1+\Delta) = f(1)$. In this case the transition operator of the process $X_\Delta(t)$ has a form of

$$T_\Delta(t)f(x) = \lambda_\Delta t f(x+\Delta) + \mu_\Delta t f(x-\Delta) + (1 - (\lambda_\Delta + \mu_\Delta))f(x) + o(t),$$

and for the infinitesimal generator after some rearrangements we get a form of

$$A_\Delta f(x) = (\lambda_\Delta + \mu_\Delta)(S_\Delta f(x) - f(x)),$$

where

$$S_\Delta f(x) = \frac{\lambda_\Delta}{\lambda_\Delta + \mu_\Delta} f(x+\Delta) + \frac{\mu_\Delta}{\lambda_\Delta + \mu_\Delta} f(x-\Delta)$$

is a transition operator, $(\lambda_\Delta + \mu_\Delta) \to \infty$ and $S_\Delta \to \mathbb{1}$ as $\Delta \to 0$. From the form of A_Δ can be seen that $\{T_\Delta(t)\}$ is really a family of pseudo-Poissonian semi-groups (see [1], page 353.) generated by A_Δ.

The elements of $\{T_\Delta(t)\}$ commute with each other too if A_Δ-s for different $\Delta > 0$ do.

$$\begin{aligned} A_{\Delta_1}A_{\Delta_2}f(x) = \lambda_{\Delta_1}(\lambda_{\Delta_2}f(x+\Delta_1+\Delta_2) + \mu_{\Delta_2}f(x+\Delta_1-\Delta_2) - (\lambda_{\Delta_2}+\mu_{\Delta_2})f(x+\Delta_1)) \\ + \mu_{\Delta_1}(\lambda_{\Delta_2}f(x-\Delta_1+\Delta_2) + \mu_{\Delta_2}f(x-\Delta_1-\Delta_2) - (\lambda_{\Delta_2}+\mu_{\Delta_2})f(x-\Delta_1)) \\ - (\lambda_{\Delta_1}+\mu_{\Delta_1})(\lambda_{\Delta_2}f(x+\Delta_2) + \mu_{\Delta_2}f(x-\Delta_2) - (\lambda_{\Delta_2}+\mu_{\Delta_2})f(x)) = \\ = A_{\Delta_2}A_{\Delta_1}f(x). \end{aligned}$$

Let $\mathcal{L}'$ be the space of the twice continuously differentiable functions defined on $[0,1]$, for which $f'(0) = f'(1) = f''(0) = f''(1) = 0$. Using a Taylor-formula for $f(x) \in \mathcal{L}'$ the infinitesimal generator has a form of

$$A_\Delta f(x) = (\lambda_\Delta - \mu_\Delta)\Delta f'(x) + (\lambda_\Delta + \mu_\Delta)\Delta^2 \frac{f''(x)}{2} + (c_1\lambda_\Delta + c_2\mu_\Delta)\frac{f''(x)}{2},$$

where the values of the constants c_1 and c_2 are

$$|c_1| \leqslant \max_{x < \xi < x+\Delta} |f''(\xi) - f''(x)| \quad \text{and} \quad |c_2| \leqslant \max_{x-\Delta < \xi < x} |f''(\xi) - f''(x)|.$$

Under the conditions of the theorem for every $f \in \mathcal{L}'$ $\sup_{0 \leqslant x \leqslant 1} |A_\Delta f(x) - A_0 f(x)| \to 0$ as $\Delta \to 0$.

Using the statement of Feller's approximation lemma our theorem is proved.

Acknowledgements

The outhor wishes to thank M. Arató (Computer and Automation Institute of Hungarian Academy of Sciences) for inspiring to solve the problem and R. Hasminskij and M. Nevelson (Institute for Problem of Information Transmission of USSR Academy of Sciences) for their advices in solving it.

References

[1] Feller W., "An Introduction to Probability Theory and its Applications" John Wiley and Sons (1970).

[2] Гихман И.И. - Скороход А.В.: Теория случайных процессов. Том 1, 11., Изд. "НАУКА" Москва, 1973.

[3] Gaver D. P., Shedler G. S., "Processor Utilization in Multiprogramming Systems via Diffusion Approximations. Ops. Res. 21(2), 569-576 (1973).

[4] Gaver D. P., Shedler G.S., "Approximate Models for Processor Utilization in Multiprogrammed Computer Systems SIAM J. Comput. 2(3), 183-192 (1973).

[5] Kobayashi H., "Application of the Diffusion Approximation to Oueueing Networks" Part I. J. ACM 21(2), 316-328 (1974)
Part II. J. ACM 21(3), 459-469 (1974)

[6] Arató M., "Diffusion Approximation for Multiprogrammed Computer Systems. An International Journal of Computers and Mathematics with Applications 1(3-4), 314-327 (1975)

Összefoglaló

Diffuziós közelités jogossága multiprogramozásu számitógépek vizsgálatában

Rét Mária

A multiprogramozásu számitógépek vizsgálatára felállitott u.n. ciklikus modellben a központi egység előtti sor eloszlására diffuziós közelitést lehet alkalmazni. A cikkben exponenciális tartásidők esetére bebizonyitjuk egyfajta diffuziós közelités jogosságát.

Р Е З Ю М Е

ОБОСНОВАННОСТЬ ДИФФУЗИОННОГО ПРИБЛИЖЕНИЯ В ОДНОЙ МОДЕЛИ МУЛЬТИПРОГРАММ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Мария Рет

В т.н. циклической модели мультипрограммной вычислительной машины применяется диффузионное приближение для вычисления распределения очереди перед центральным процессором.

В докладе доказывается справедливость одного диффузионного приближения в случае показательного распределения времен обслуживания.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

ON THE SPEED OF COMPUTERS WITH PAGED AND INTERLEAVED MEMORY

Iványi A. – Kátai A.

Abstract:

A performance measure (the speed) of computer mathematical models is defined. This measure is given as a function of hardware and program behaviour for Bélády's computer model with paged memory and Volihiman's model with interleaved memory.

KEY WORDS AND PHRASES: computer systems performance, demand paging, interleaved memory behaviour.

1. Intruduction

Computer performance is investigation by empirical, simulation and analytical methods [1].

Tha analytical methods is based on the analysis of mathematical by "exact" methods (e.g. queueing or Markov-chain theory, compinatorics etc.).

Due to the inacuuuracy of models the analytical method usuallly gives only a rough estimate, but the results are general and convenient for computer plannning or development.

In this lecture we recommend and analytical method, based on Bélady's[2], Coffman's [3] and Kogan's [4] methods and give some concrete formulas derived by this method.

2. Definition of the speed

The set $N = \{ \nu_1, \ldots, \nu_n \}$ $(1 \leqslant n < \infty)$ is called a program, and the sequence $\omega_T = r_1 \ldots r_T$ $(1 \leqslant T \leqslant \infty, r_t \in N, t = 1, \ldots, T)$ consisiting of elements of N. (T-elements permutations with repetition) is called a program realization of lenght T. Denote N^T the set of all possible sequences ω_T. Denote $\tau[\omega_T]$ the processing time of a sequence ω_t on given computer model. The distribution of the elements of N in the sequence ω_T is called program behaviour [5]. This behaviour is given by the set $D = \{ D_1, \ldots, D_T \}$ of distribution function $D_1, \ldots, D_T$ where $D_T[\omega_T]$ gives the probability of ω_T in the space of events N^T, that is

(2.1.) $\forall \omega_T \quad 0 \leqslant D_T[\omega_T] \leqslant 1$

and

(2.2.) $\forall_T \quad \sum_{\omega_T \in N^T} D_T[\omega_T] = 1.$

Further we supppose

$$(2.3) \qquad \sum_{i=1}^{n} D_{T+1}[\omega_T \nu_i] = D_T[\omega_T].$$

Instead of $D_T[\omega_T]$ we use the marking $D[\omega_T]$. Denote the set of D's satisfying the conditions (2.1), (2.2) and (2.3) by D.

In this lecture we use 6 simple behaviour model: homogeneous [6], cyclical [6], random [2], random with step [3], random with repetition [7] nad independent [5] ones, Let HOM, CYCL, RAN, STEP_p, REP_p and $\mathrm{IND}_{p\,1,\ldots,p_n}$ denote then.

According to the homogeneous model the references are equivalent, that is

$$(2.4) \qquad P\{r_1 = \nu_i\} = \frac{1}{n} \quad \text{and} \quad r_t = r_1 \qquad (t = 2,3,\ldots : \quad i = 1,\ldots,n).$$

This formula is equivalent to the following definition:

$$(2.5) \qquad \mathrm{HOM}'[\omega_k] = \begin{cases} \frac{1}{n}, & \text{if in} \quad \omega_k \quad r_1 = r_2 = \ldots = r_k \\ & \qquad\qquad\qquad\qquad (k = 1,2,\ldots). \\ 0, & \text{otherwise.} \end{cases}$$

According to the cyclical model the step ν_i, ν_{i+1} $(\nu_{n+1} \equiv \nu_1)$ has a probability 1, that is

$$(2.6) \qquad P\{r_1 = \nu_i\} = \frac{1}{n} \quad \text{and} \quad P\{r_{t+1} = \nu_{i+1} \mid r_t = \nu_i\} = \begin{cases} 1, & \text{if} \quad r_t = \nu_i, \\ 0, & \text{if} \quad r_t \neq \nu_i \end{cases}$$

$$(t = 1,2,\ldots; \quad i = 1,\ldots,n).$$

This formula is equivalent to the following definition:

$$(2.7) \qquad \mathrm{CYCL}[\omega_k] = \begin{cases} \frac{1}{n}, & \text{if in} \quad \omega_k \quad \text{from} \quad r_t = \nu_i, \quad r_{t+1} = \nu_j \\ & \qquad\qquad \text{follows} \quad j \equiv i + 1 \pmod{n} \\ 0, & \text{otherwise.} \end{cases}$$

$$(t = 1,2,\ldots).$$

According to the random model the references occur randomly, that is

$$(2.8) \qquad P\{r_t = v_i\} = \frac{1}{n} \qquad (t = 1,2,\ldots; \quad i = 1.,\ldots,n).$$

This formula is equivalent to the following definition:

$$(2.9) \qquad \mathrm{RAN}[\omega_k] = \frac{1}{n^k} \qquad (k = 1,2,\ldots, \omega_k \epsilon N^k).$$

According to the random model with repertition the repertition has a probability p, and other references have a probabiliy $\frac{1-p}{n-1}$:

$$(2.10) \qquad P\{r_1 = v_i\} = \frac{1}{n}, \quad P\{r_t = v_i\} = \begin{cases} p, & \text{if} \quad r_t = v_i, \\ \frac{1-p}{n-1} & \text{if} \quad r_t \neq v_i, \end{cases}$$

$$(t = 2,3,\ldots; \quad i = 1,\ldots,n)$$

This formula is equivalent to the following definitiom:

$$(2.11) \qquad \mathrm{REP}_p[\omega_k] = \frac{1}{n} \cdot p^f \left(\frac{1-p}{n-1}\right)^{k-f-1} \qquad (k = 1,2,\ldots),$$

where f is the number of the repertitions in ω_k.

Acccording to the random model with step [3] the step v_i, $v_{i+1}(v_{n+1} \equiv v_1$ in ω_k has a probability p, and other references have a probability $\frac{1-p}{n-1}$:

$$(2.12) \qquad P\{r_1 = v_i\} = \frac{1}{n}; \; P\{r_t = i+1\} = \begin{cases} p, & \text{if} \quad r_{t-1} = v_i, \\ \frac{1-p}{n-1}, & \text{if} \quad r_{t-1} = v_i, \end{cases}$$

$$(t = 2,3,\ldots; \quad i = 1,\ldots,n).$$

This formula is equivalent to the following definition:

$$(2.13) \qquad \mathrm{STEP}_p[\omega_1] = \frac{1}{n}; \qquad \mathrm{STEP}[\omega_k] = \frac{1}{n} \cdot p^f \left(\frac{1-p}{n-1}\right)^{k-f-1} \qquad (k = 1,2,\ldots),$$

where f is the number of the steps in ω_k.

According to the independent model [5] the reference to the page ν_i has a probability p_i, that is

(2.14) $$P\{r_t = \nu_i\} = p_i \quad (t = 1,2,\ldots).$$

This formula is equivalent to the following definition:

(2.15) $$\mathrm{IND}_{p_1,\ldots,p_n}[\omega_k] = \prod_{i=1}^{n} (p_i)^{f_i},$$

where f_i is the number of the references to the page ν_i.

Computer performance is characterized by the number of operations in a time unit: V.V is called the speed of the computer model and is determined by the formula

(2.16) $$V \overset{\text{def}}{=} \lim_{k \to \infty} \inf \frac{1}{\sum'_{\omega_k \in N^k} D[\omega_k] \frac{\tau[\omega_k]}{k}}.$$

If in (2.16) we have existance of the lim in addition to the lim inf, then this limit is denoted by V'.

Our aim is to determine the speed for various computer and program behaviour models.

3. The mathematical model of computers with paged memory

For the investigation of computers with paged memory we use the well-known model proposed by Bélády [2] in 1966.

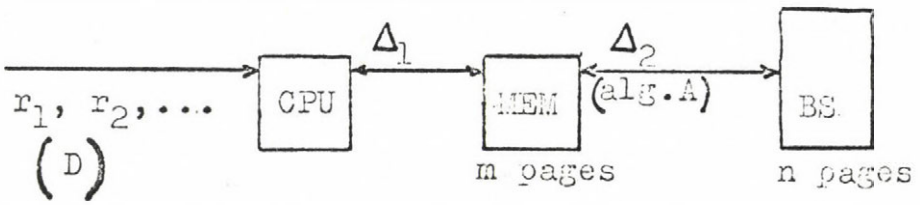


Fig. 1. The scheme of a computer with 2 level paged memory

The computer consists of a central processor unit (CPU),– an m-paged main memory (MEM) and an n-paged backing store (BS). The CPU has direct access to MEM-access time Δ_1 – while an indirect access to BS-access time $\Delta_1 + \Delta_2$. The paging is controlled by a demand paging algorithm. The set of demand paging algorithms by A

For this model speed V_p is [8]

(3.1) $$V_p = \frac{1}{\Delta_1 + \Delta_2 \cdot C},$$

where C is the average cost of a reference, that is the page fault probability [5]. By definition

(3.2) $$C = C(m,n,A,D) = \limsup_{k\to\infty} \sum_{\omega_k \in N^k} D[\omega_k]\left(\frac{\sum_{i=1}^{k} \delta_i}{k}\right)$$

where $A \in A$, $D \in D$,

(3.3) $$\delta_i = \delta(i,m,n,\omega_T,A) = \begin{cases} 0, & \text{if } r_i \in S_t, \\ 1, & \text{if } r_i \notin S_t, \end{cases}$$

and S_t is the set of pages in MEM at time t. S_t is the set of pages in MEM at time t. S_t is called the memory state. If in (3.2) there exist a limit, then it is denoted by C'.

4. General assertions on the speed of computers with paged memory

Lemma 1. ([7]) *If* $\Delta_1 > 0$, *and* $1 \leqslant m \leqslant n < \infty$, *then*

(4.1) $$0 = C(m,n,A,\text{HOM}) \leqslant C(m,n,A,D) \leqslant C(m,n,\text{LRU,CYCL}) = 1,$$

that is for the speed

(4.2) $$\frac{1}{\Delta_1 + \Delta_2} = V_p(\Delta_1,\Delta_2,m,n,\text{LRU,CYCL}) \leqslant V_p(\Delta_1,\Delta_2,m,n,A,D) \leqslant$$

$$\leqslant V_p(\Delta_1,\Delta_2,m,n,A,\text{HOM}) = \frac{1}{\Delta_1}$$

holds.

Definition. 1. ([9]). *The demand paging algorithms, for which*

(4.3) $$\forall T_1, \forall T_2 \quad \sum_{i=1}^{T_1} \delta(i,m,n,\omega_{T_1},A) = \sum_{i=1}^{T} \delta(i,m,n,\omega_{T_1}\omega_{T_2},A)$$

are called sequential [6]. *The set of the sequential algorithms is denoted by* B.

Lemma 2. *If* $1 \leqslant m \leqslant n < \infty$, *then for every* $B \in B$ *and for every* $D \in D$

(4.4) $$C_{\inf} = \liminf \sum_{\omega_k \in N^k} D[\omega_k]\delta_k \leqslant C(m,n,B,D) \leqslant \limsup_{k\to\infty} \sum_{\omega_k \in N^k} D[\omega_k]\delta_k.$$

Definition 2. Let $D \epsilon D$ and N_+^k $(k = 1,2, \ldots, ; N_+^k \subseteq N^k)$ be given. Denote a_k the sum $\sum_{\omega_k \epsilon N_+^k} D[\omega_k]$. If

$$\lim_{k \to \infty} a_k = 0, \tag{4.5}$$

then we shall say, that the sequence N_+^k has zero limitdensity in N^k.

Lemma 3. ([7]). *If for a given D there exist an m-tuple of pages $\mu_1, \ldots, \mu_m$ and $\mu_1, \ldots, \mu_m$ and $\epsilon > 0$, for whicg*

$$\forall \omega_k \quad D[\omega_k \nu_i] \geqslant \epsilon . \quad D[\omega_k] \tag{4.6}$$

holds, then the sequence N_+^k has zero limit-density in N^k, where N_+^k is the set of $\omega_k \epsilon N^k$, for which $|S_t| = |S_t(m,\omega_k,B)| < m$.

Definition 3. Let ω_T be given. The sequences of lenght $(T + f)$ $(f = 0,1, \ldots)$ identical to ω_T up to the T-th element, are called the bundle $\Pi_f[\omega_T]$ with root ω_T and lenght f.

Definition 4. The average cost of a references C^{Π} in a given bundle $\Pi_f[\omega_T]$ is by definition

$$C = C^{\Pi}(m,n,A,D,\omega_T) = \lim_{k \to \infty} \sum_{\omega_k \epsilon \pi_{k-T}[\omega_T]} D[\omega_k]\delta_k , \tag{4.7}$$

where $D^{\Pi}[\omega_k]$ is the probability of sequence ω_k $(\omega_k \epsilon \Pi_{k-T}[\omega_T])$ within the bundle, that is

$$D^{\Pi}[\omega_k] = \frac{D[\omega_k]}{\sum_{\omega_k \epsilon \pi_{k-T}[\omega_T]} D[\omega_k]} . \tag{4.8}$$

Lemma 4. ([7]). *Let N_+^k denote the set of ω_k' not belonging to any bundle, which has a cost C^{Π}. If the sequence N_+^k, then $C(m,n,B,D)$ exists and is C^{Π}.*

5. Theorems on the speed of computers with paged memory

Theorem A. (*Belady*, 1966) [2]. *If L is a nonlookahead demand paging algorithm, then*

$$C(m,m,L,\mathrm{RAN}) = \frac{n-m}{n} . \tag{5.1}$$

Theorem B. (*Aho, Denning, Ullman* 1971/([5]). *If* $1 \leqslant m \leqslant n < \infty$.

then

$$(5.2)\qquad C'(m,n,\mathrm{OPT},\mathrm{IND}) = \sum_{i=m}^{n} p_i - \frac{\sum_{i=m}^{n} p_i^2}{\sum_{i=m}^{n} p_i},$$

where OPT *is the optimal paging algorithm, always replacing the page of* S_t *with minimal* p_i[5].

Theorem C. (*Stoyan*, 1975/([8]). *If* $1 \leqslant m \leqslant n < \infty$, *then*

$$(5.3)\qquad C'(m,n,\mathrm{REF}_a,\mathrm{RAN}) = \frac{n-m}{n+a} \quad (a = 0,1,\ldots,m-1),$$

where REF_a *is a lookahead algorithm, which knows a references ahead, and holds required pages in the memory if possible, and chooses randomly among the others.*

Theorem 1. ([7]). *If* $1 \leqslant m \leqslant n < \infty$, *and* $0 \leqslant a \leqslant m$, *then*

$$C'(m,n\mathrm{REF}_a,\mathrm{REP}_p) =$$

$$(5.4)\qquad \frac{(n-m)(1-p)}{n-1+[\min/a,m-1/]1-p; + [\max/0,a-m+1/]\left[1-m!\,\frac{/1-p/^{m-1}}{/n-1/^{m-1}}\right](1-p)}$$

Theorems A and B follow from theorem 1 (in cases $a = 0$, $p = \frac{1}{n}$ and $0 \leqslant a \leqslant m-1$, $p = \frac{1}{n}$).

Theorem 2. ([7]). If $1 \leqslant m \leqslant n < \infty$, then

$$(5.5)\qquad C'(m,n,PP_b,\mathrm{RAN}) = \frac{n-m}{n}\left(\frac{n-1}{n}\right)^b \qquad (b = 0,1,),$$

where PP_b is a lookahead algorithm, which knows at time t the next b references, differing from r_t and each other, and hold these pages if possible, in the memory, and chooses randomly among the others.

In case $b = 1$, $m = 2$, $n = 3$, it follows from theorem 2. the partial resolution of the problem, investigated by Bélády in 1966, namely $C'(2,3,\mathrm{MIN},\mathrm{RAN}) = \frac{2}{9}$.

Theorem 3. ([7]). If $a \geqslant 0$, then

$$C'_{(2,3,\mathrm{REF}_a,\mathrm{REP}_p)} =$$

(5.6)
$$= \frac{1-p}{2 + [\min/a, 1/]/1 - p/ + \operatorname{sign}[\max/0, a - 1/]p/1 - p^{a-1}/}$$

From this theorem it follows (in the case $a \to \infty$, when $\mathrm{REF}_a \to \mathrm{MIN}$), that

(5.7) $$C'(2,3,\mathrm{MIN},\mathrm{REP}_p) = \frac{1-p}{3} .$$

6. Mathematical model of computers with interleaved memory

We investigate the following model of computers with interleaved memory due to V.E.Vulihman [10]:

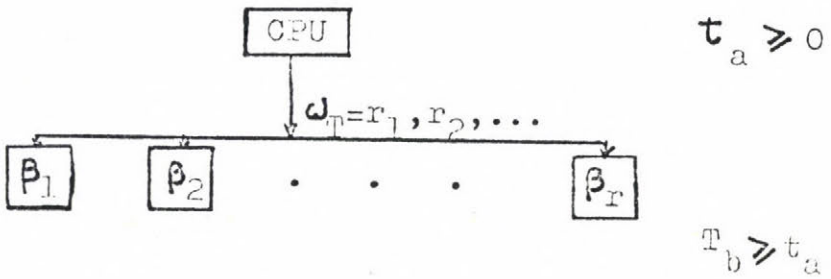


Fig. 2. Scheme of computers with interleaved memory

The computer consists of a central processor unit (CPU) and models of memory $\beta_1, \ldots, \beta_r$. The set of modules is denoted by B. The elements of ω_T are genoted by the CPU requiering $t_a \geqslant 0$ time per element. A request to a module reserve that moduöe for a time $T_b > t_a$ during this time other request can't be served by this module. If the module being request is occupied, then the generation of ω_T will be susepended until the modul is free.

The speed of this modul is denoted by V_i.

7. General assertions on the spead of computers with interleaved memory

Hellerman in his book [6], Bokova and Tzaturyan in their paper [11] proved the following assertions.

Lemma 5. ([6]). *If* $T_b > t_a \geqslant 0$, *then for every* $D \in D$ *and for every* $r \geqslant 1$

(7.1) $$\frac{1}{T_b} = V_i'(t_a, T_b, r, \mathrm{HOM}) \leqslant V_i(t_a, T_b, r, D) \leqslant V_i(t_a, T_b, r, \mathrm{CYCL}) = \frac{1}{T}\min(r, \frac{T_b}{t_a}),$$

where HOM and CYCL are the homogeneous and cyclical bahaviour models.

Definition 5. *Let* $\rho_i(\rho_0 = 0)$ *denote the processing time of* ω_T *up to i-th element. Then the increment due to the i-th element is* $\sigma_i = \rho_i - \rho_{i-1}$.

Example. For every $\omega_2 \epsilon N^2$

$$(7.2) \qquad \sigma_1 = T_b + t_a, \sigma_2 = \begin{cases} T_b, & \text{if} \quad r_2 = r_1 \\ t_a, & \text{if} \quad r_2 \neq r_1 \end{cases}$$

Lemma 6. ([11]). *If* $T_b > t_a \geqslant 0$, *then*

$$(7.3) \qquad \operatorname*{Lim\,inf}_{k\to\infty} \frac{1}{\sum_{\omega_k \epsilon N^k} D[\omega_k]\sigma_k} \leqslant V_i(t_a, T_b, r, D)$$

$$\leqslant \limsup_{k\to\infty} \frac{1}{\sum_{\omega_k \epsilon N^k} D[\omega_k]\sigma_k}$$

Lemma 7. ([11]). *If* $T_b > t'_a > t''_a \geqslant 0$, *then*

$$(7.4) \qquad V_i(t'_a, T_b, r, D) \leqslant V_i(t''_a, T_b, r, D).$$

Lemma 8. ([11]). *If* $T_b > t_a \geqslant 0$, *and* $r' > r''$, *then*

$$(7.5) \qquad V_i(t_a, T_b, r', D) \geqslant V_i(t_a, T_b, r'', D).$$

8. Theorems on the speed of computers with interleaved memory

Theorem 4. ([7]). If $T_b > t_a \geqslant 0$, then for $r \geqslant 1$ $V'_i(t_a, T_b, r, \mathrm{RAN}) \leqslant$

$$(7.6) \qquad \leqslant \frac{1}{t_a \sum_{i=1}^{r} \sum_{j=2}^{i} p_{i,j} + \left(\sum_{i=1}^{r} p_{i,1}\right)\left[\sum_{k=0}^{r-1} (\max(t_a, T_b - k \cdot t_a) \sum_{j=k+1}^{r} \frac{p_j}{j})\right]}$$

and the equality holds

a.) if $t_a = 0$, then for $r = 1$;

b.) if $\frac{T_b}{2} \leqslant t_a \leqslant T_b$, then for $r \geqslant 1$;

c.) if $0 < t_a < \frac{T_b}{2}$, then for $r = 1,2$.

In the formula (7.6)

$$p_i = \frac{r(r-1)\ldots(r-i+1)}{r^{i+1}}\, i \quad (1 \leqslant i \leqslant r), \tag{7.7}$$

and

$$p_{i,j} = \frac{p_i}{\sum_{i=1}^{r} i \cdot p_i} \quad (1 \leqslant i \leqslant r, \quad 1 \leqslant j \leqslant i). \tag{7.8}$$

The following corollaries foolow from theorem 4. as special cases.

Corollary 1 (Hellerman, 1967.) ([6]). If $t_a = 0$ and $r \geqslant 1$, then

$$V_i'(0,T_b,r,\mathrm{RAN}) = \frac{}{T_b} \sum_{i=1}^{r} i \cdot p_i. \tag{7.9}$$

Burnett, Coffman [3] and Stone [12] proved a more general assertion.

Theorem D. (Burnett, Coffman, Stone 1974.) [3.12].

If $t_a = 0$ and $r \geqslant 1$, then

$$V_i'(0,T_b,r,\mathrm{STEP}_p) = \tag{7.10}$$

$$= \frac{2}{T_b} \sum_{k=1}^{r} \sum_{j=0}^{k-1} (k-1)_p\, j \left(\frac{1}{n-1}\right)^{k-j-1} \cdot C_{n-j,k-j},$$

where

$$C_{n,k} = \sum_{j=0}^{k-1} [(-1)^j \binom{k-1}{j} (n-j-1)(n-j-2)\ldots(n-k+1)]. \tag{7.11}$$

Corollary 2. If $\frac{T_b}{2} \leqslant t_a \leqslant T_b$ and $r \geqslant 1$, then

$$V_a'(t_a,T_b,r,\mathrm{RAN}) = \frac{1}{\frac{1}{r}T_b + (1-\frac{1}{r})t_a}. \tag{7.12}$$

Corollary 3. If $T_b > t_a \geqslant 0$, then

$$(7.13) \qquad V_b'(t_a,T_b,2,\mathrm{RAN}) = \frac{1}{\frac{1}{3}t_a + \frac{1}{2}T_b + \frac{1}{6}\max(t_a, T_b - t_a)}.$$

On the base of the formula (7.9) it is not easy to estimate the order of $V_i'(0,T_b,r,\mathrm{RAN})$, therefore the following theorems are interesting.

Theorem E. (Hellerman, 1967)([6]). If $1 \leqslant r \leqslant 45$, then

$$(7.15) \qquad 0{,}96 r^{0{,}56} \leqslant V_i'(0,T_b,r,\mathrm{RAN}) \leqslant 1{,}04 \cdot r^{0{,}56}.$$

Theorem F. (Vulihman, 1972.)([10]). If $r \geqslant 1$, then

$$(7.16) \qquad V_a'(0,T_b,r,\mathrm{RAN}) \leqslant \sqrt{2\Pi r}.$$

We proved the following more general theorems.

Theorem 5. ([13]). If $t_a = 0$ and $r \geqslant 1$, then

$$(7.17) \qquad \frac{1}{T_b}\left(\sqrt{\frac{\pi r}{2}} - 1\right) < V_i'(0,T_b,r,\mathrm{RAN}) = \frac{r!\sum_{k=0}^{r-1}\frac{r^k}{k!}}{T_b \cdot r^r} \frac{1}{T_b}\left(\sqrt{\frac{\Pi r}{2}} + 1\right).$$

In our paper [7] we used a simple direct proof. Using a result due to G. Szegő [14] we can proof a formula with a smaller additive constant, which is exact.

Theorem G. (Szegő, 1928)([14]). If q is a nonnegative integer number, then

$$(7.18) \qquad \frac{1}{2}e^q = 1 + \frac{q}{1!} + \frac{q^2}{2!} + \ldots + \frac{q^q}{q!}\Theta_q,$$

where $\Theta_0 = \frac{1}{2}$ and Θ_q thends monotonically to $\frac{1}{3}$ as $q \to \infty$.

Theorem 6. If $t_a = 0$ and $r \geqslant 1$, then

$$(7.19) \qquad V_i'(0,T_b,r,\mathrm{RAN}) = \frac{1}{T_b}\left(\sqrt{\frac{\Pi r}{2}} - \frac{1}{3} + \rho_r\right),$$

where ρ_r tends monotonically to zero as $r \to \infty$ and

$$(7.20) \qquad \rho_1 = \frac{4}{3} - \sqrt{\frac{\Pi}{2}} \approx 0{,}08 \quad \text{and} \quad \rho_2 = \frac{11}{6} - \sqrt{\Pi} \approx 0{,}06.$$

It seems a hard but resolvable problem to estimate the order of expression in Coffman's theorem, as a function of p.

References

[1] U. Grenander, R.F. Tsao, Ouantitative methods for evaluating compuetr system performance: review and propsal (in Freiberger W., Statistical compter performance evaluation, Academic Press, New York, 1972, 3-24).

[2] L.A. Bélády, A study of replacement algorithms for a virtual storage compuetr, IBM Systems Journal, 5. No. 2. (1966), 282-288.

[3] G.J. Burnett, E,G. Jr. Coffman, A combinattorial problem related to interleaved memory systems, J. of ACM, 20, No. 1. (1973), 39-45.

[4] Коган Я.А., Марковские модели управления обменом в двухуровневой памяти ЦВМ, Автоматика и телемеханика, 28, Но. 4. /1973/ 146-154.

[5] A.V. Aho, P. J. Denning, Principles of optimal page replacement, J. of ACM, 18. No. 1. (1971), 80-93.

[6] H. Hellerman, Digital computer principles, McGraw Hill, New York, (1967).

[7] Ивани А., Исследование скорости ЭВМ со страничной и блочной организацией памяти, Кандидатская диссертация, Московский Государственный Университет, 1975.

[8] E.G. Jr. Coffman, P.J. Denning, Operating Systems Theory Englewood Cliffs, Prentice Hall, (1973).

[9] Стоян Ю.А., Оценка эффективности алгоритмов замещения, Программирование, 1, Но. 1. /1975/, 22-25.

[10] Вулихман В.Е., Исследование методов организации оперативной памяти, Кндидатская диссертация, Москва, 1972.

[11] Бокова Е.Е., Цатурян Г.К., Сопоставление известных определений скорости ЭВМ с блочной памятью, Конкурсная работа, Московский Государственный Университет, 1975, 1-14.

[12] H.S. Stone, A note on a combinatorial problem of Burnett and Coffman, Com. of ACM. 17. No. 3. (1974), 165-166.

[13] A. Iványi, I. Kátai, Lowee and upper estimates for speed of computers with blocked memory, Preprint of Eötvös L. University, Budapest, 1975. 1-14.

[14] G. Szegő, Uber einige von S. Ramanujen gestellte Aufgaben, J. of the London Mathematical Society, 3. Part 3. (1928), 225-232

Összegoglaló

Lapozott és atlapolt memóriájú számitógépek sebessége

Iványi A. – Kátai I.

Az előadásban definiáljuk a számitógépek teljesitményét jellemző mennyiséget, a sebességet. Ezt a sebességet megadjuk a hardwareő és a programviselkedési paraméterek függvényében a lapozott memóriáju számitógépek Béládу-féle és az átlapolt memóriáju számitógépek Vulihman-féle modelljére.

Kulcsszavak és kifejezések: számitógépek teljesitménye, igénye szerinti, lapozás, átlapolt memória, programviselkedés.

Р Е З Ю М Е

О скорости ЭВМ со страничной и блочной памятью

А. Ивани и И. Катаи

Резюме: определена мера /скорость/ производительности математических моделей ЭВМ и задана эта скорость, как функция параметров аппаратуры и поведения программ для математической модели ЭВМ со страничной памятью /предложенной Л.А. Белади/ и математической модели ЭВМ с блочной памятью /предложенной В.Е. Вулихманом/.

Ключевые слова и выражения: производительность вычислительных систем, страничная память, блочная память, поведение программ.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

PARALLEL FOLYAMATOK GRÁF MODELLJEI

Szlankó János
MTA Központi Fizikai Kutató Intézet

1. Bevezetés

A programozás alapvető problémája a programok korrektségének a bizonyitása. Ez különösen nehéz feladatot jelent akkor, ha egyes programrészek párhuzamos feldolgozása is megengedett, sőt szükséges mint például operációs rendszerekben, real-time rendszerekben. Szükséges egy preciz és lehetőleg szemléletes elméleti eszköz, amelynek segitségével az ilyen rendszerek konstuálása megkönnyithető, specifikáció szerinti müködésük ellenőrizhető. A korábbiakban a konstruálás valamilyen ad hoc módon történt, a helyes müködést pedig teszt feladatok elvégzésével valószinüsitették. A 60-as évek végén jelentek meg a Dijkstra féle P és V operációk. Ezek alkalmazásával már lehetett kérdéseket feltenni és megoldani a rendszer korrektségével kapcsolatban, de a bizonyitások kissé nehézkesek. Az utóbbi években kezdik alkalmazni a Petri hálókat parallel folyamatok modellezésére. A Petri hálók segitségével egy parallel feldolgozható programrendszernek olyan tulajdonságait (kommunikációs, szinkronizációs) állapithatjuk meg, amelyek függetlenek a részfolyamatok végrehajtási idejétől, tervezésnél pedig felhasználhatók arra, hogy a megvalósitandó programrendszernek bizonyos tulajdonságai függetlenek legyenek a részfolyamatok végrehajtási idejétől tehát a kontrol szerkezet biztositsa, tekintet nélkül a processzor ütemezési stratégiájára). Az alábbiakban a Petri hálók elméletéből ismertetünk néhány hasznos fogalmat, eljárást és alkalmazást.

2. Petri hálók

Egy *Petri háló* egy négyes $N = (P,T,per,post)$, ahol P és T véges diszjunkt halmazok, *pre* és *post* pedig bináris relációk:

$$pre \in PXT$$
$$post \in PXT$$

P és T minden elemének szerepelni kell legalább egy reláció elemben. P elemeit *helyek*nek, T elemeit *tranziens*eknek nevezzük. $p_1 \epsilon P$ input helye a $t \epsilon T$-nek, ha $(p_1,t) \epsilon\ pre$, $p_2, \epsilon P$ output helye $t \epsilon T$-nek, ha $(p_2,t) \epsilon\ post$.

Egy Petri hálót szemléletesebben megadhatunk egy irányitott páros gráffal. A helyeket körökkel, a tranzienseket vonással jelöljük. Ha $(p_1\ t) \epsilon\ pre$, akkor p_1-ből vezet egy él t-be, ha $(p_1,t) \epsilon\ post$, akkor t-ből vezet egy él p_1-be.

Ugyanezt a Petri hálót számitási célokra mátrox formában is megadhatjuk, ha a háló *tisz-*

ta : nincs olyan hely, amely egy tranziensnek az input és output helye is. Egy N Petri hálóhoz tartozó C incidencia mátrixot a következőképpen definiáljuk: a sorokat a P halmaz elemeivel, az oszlopokat a T halmaz elemeivel indexeljük és

$$C(p,t) = \begin{cases} -1, & \text{ha } (p,t) \in pre \\ +1, & \text{ha } (p,t) \in post \\ 0 & \text{egyébként} \end{cases}$$

Az 1. ábra hálóját a következő mátrixszal adhatjuk meg:

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{array} \begin{array}{c} \begin{array}{ccc} t_1 & t_2 & t_3 \end{array} \\ \begin{pmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix} \end{array}$$

Egy $M : P \to I$ függvényt az N háló pontozásának nevezzük. I a nemnegativ egészek halmaza. Ha $p \in P$, akkor $M(p)$ a p helyen lévő pontok száma. Egy helyet pontozottnak nevezünk, ha $M(p) \geqslant 1$. Az M függvényt egy vektorral adhatjuk meg, ahol a vektor elemeit a helyekkel indexelhetjük. A (N,M) párt *pontozott Petri hálónak* nevezzük.

Egy $t \in T$ tranziens *aktivált* egy adott M pontozás alatt, ha $M \geqslant (B(-,t))$ ahol

$$B(p,t) = \begin{cases} 1, & \text{ha } p \text{ input helye } t\text{-nek} \\ 0 & \text{egyébként} \end{cases}$$

A fenti egyenlőtlenség azt jelenti, hogy a hálóban legalább t input helyei pontozottak. Egy aktivált $t \in T$ tranziens *átbillenése* az M pontozást egy M_1 pontozásba transzformálja ugy hogy $M_1 = M + C(-,t)$. Ez az jelenti, hogy a t input helyein a pontok száma eggyel csökken és minden output helyén eggyel megnő a pontok száma.

A háló *kiértékelése* során az aktiv tranziensek közül (ha van egyáltalán) egy tetszőleges kiválasztásra kerül, átbillen. Ezzel előáll egy új pontozás, amelyen folytatódhat a kiértékelés.

3. A Petri hálókkal kapcsolatos legfontosabb kérdések

Ha egy M kezdeti pontozású háló minden p helyére igaz, hogy a háló müködése során p helyen lévő pontok száma egy k korlátnál nem lehet nagyobb, akkor a háló *k-biztonságosnak* mondjuk. Ez azt jelenti, hogy az M-ből elérhető pontozások $[M]$ halmaza véges. (M_i pontozásból M_j *elérhető*, ha a kiértékelés során van a billenések olyan S sorozata, amely az M_i pontozást M_j-ben viszi át). A biztonság kérdésének eldönthetősége nagyon fontos a további vizsgálatok szempontjából. Szerencsére a kérdés eldönthető [5]. A Petri hálók részosztályai-

ra vonatkozó biztonsággal foglalkozik [8], [9], [10]. A továbbiakban csak biztonságos hálókkal foglalkozunk. A gyakorlatban előforduló hálók biztonságosak, vagy viszonylag egyszerü módositással funkciójuk megzavarása nélkül biztonságossá tehetők.

Egy (N,M) hálóhoz hozzárendelhető egy *tranziens-gráf*, amelyet fontos kérdések megválaszolásához használhatunk fel. A tranziens gráf irányitott gráf, amely a következőképpen konstruálandó.

A szögpontokat az $[M]$, az éleket T elemeivel fogjuk indexelni. Legyen a konstruáláshoz a kezdőpont M. Az M esetben aktivált összes tranziensre nézzük meg, hogy milyen pontozásba viszi át M-t, és a különböző pontozásokhoz rendeljünk különböző szögpontokat, mindegyikbe élt irányitva M-ből, az éleket a megfelelő tranzienssel indexelve. Válasszunk ki egy másik M_1 szögpontot (pontozást) és ismételjük meg az eljárást. Folytassuk ezt addig, amig minden – a közben keletkező szögpontokra is – el nem végeztünk. A konstruálás biztonságos gráfoknál nyilván mindig véges lépés után végetér. Példaként nézzük a 2. ábrát.

A gráf alapján az elérhetőség is eldönthető: M_i pontozásból M_j elérhető, ha van olyan irányitott út, amely M_i-ből M_j-be vezet.

Egy M_i *holt pontozás,* ha nincs aktivált tranziens. Ez a gráfban azt jelenti, hogy M_i-ből nem vezet ki él. Egy M_i pontozás *életképes,* ha nem mehet át holt pontozásba.

Egy M_i pontozás *reprodukálható* ha van a billenéseknek olyan sorozata, amely önmagába viszi át. Egy $[M]$ reprodukálható, ha minden eleme reprodukálható. A tranziens gráfban ez azt jelenti, hogy M_i egy irányitott körön helyezkedik el, illetve a gráf erősen összefüggő.

Az $M_j = M_i + CX$ egyenletrendszer nemnegativ egész megoldásaiban egy $t \epsilon T$ tranziensre $X(t)$ azt jelenti, hogy t-nek hányszor kellene billeni ahhoz, hogy M_i átmenjen M_j-be. Az hogy egy X megoldásvektor realizálható-e a tranziesgráf alapján eldönthető. Ha C' jelenti C transzponáltját, a $C'.\ z = 0$ egyenlet nemnegativ egész megoldásait *invariánsoknak* nevezzük, ugyanis

$$M_j'z = (M_i + CX)' \cdot z = M_i'z + (CX)' \cdot z =$$
$$= M_i'z + X'C'z = M_i'z \quad \text{ha} \quad C'z = 0$$

Ha a z vektor minden eleme a $\{0,1\}$ halmazból kerül ki, akkor $M'z$ egyenlő azon pontok összegével, amelyek azon a helyeken találhatók amelyekhez 1 tartozik a z vektorban. Innen származik az invariáns elnevezés: ez az összeg a háló müködése során nem változik.

4. A hálók alkalmazása

a.) Ha a helyeknek feltételeket, a tranzienseknek eseményeket (az események pillanatszerüek) feleltetünk meg, akkor az olyan folyamatok, amelyek csak induláskor és befejezéskor lépnek kapcsolatba a környezetünkkel méghozzá úgy, hogy az indulás feltételektől függ, befejezéskor pedig feltételeket állitanak be feltétel-esemény szempontjából a 3.a. szerint modellezhetők.

Petri hálókkal csak, olyan folyamatokat modellezhetünk, ahol a feltétel-esemény szerkezet időben nem változik.

b.) Mivel a program feldolgozása is folyamat, ezen a téren is alkalmazhatjuk a Petri hálókat. A program alapján létrehozhatunk egy hálót: feleltessünk meg a program változóit számára történő értékadásnak a 3.a. ábrát (egy be- és egy kimeneti feltétellel), a feltételes elágazásnak a 3.b. ábrát, és ha program feltétel változókon való müveletként tartalmazza a P és V operációkat [1], [2], akkor azoknak a 3.c. ábrát. Az igy kapott háló nem tartalmaz információt a müveletek, predikátumok interpretációjáról, tehát a program olyan tulajdonságai vizsgálhatók, amelyek bármilyen interpretációt mellett igazak. (Az interpretációra vonatkozó ismeretek is bevihetők a hálóba, de ennek korlátai vannak: Petri hálókkal nem valósitható meg minden kiszámitható függvény [5].) A 4. ábrán látható egy program, amely a feldolgozandó adatokat egy bufferban kapja és egy másik bufferbe helyezi az eredményt. Mellette látható egy háló, amelyet a fenti szabályok szerint képeztünk. A hálón a vizsgált kérdéstől függően egyszerüsitések végezhetők: ha például a tranziensgráfot csak a holtpont keresésre akarjuk felhasználni, a háló több eleme összevonható.

c.) Ha már adott a folyamat hálója és a kezdeti pontozás a tranziens gráfon vizsgálhatók a holtpont, kölcsönös kizárás stb. A kölcsönös kizárást az invariánsok segitségével kényelmesebb lehet, mert igy nem kell az esetleg negyméretü gráfot létrehozni. (A lineáris algebra még számos, a Petri hálók alosztályait érintő kérdéshez felhasználható [9]).

Nézzük az 5. ábra hálóját. A háló egyik invariánsa $z' = (1\ 0\ 1\ 1\ 0)$ vektor. Ha a kezdeti pontozás az ábrán kijelölt vagyis $M' = (1\ 1\ 0\ 0\ 1)$, akkor $M'z = 1$. Ez azt jelenti, hogy P_1, P_3, P_4 helyek közül mindig csak egy tartalmaz 1 pontot.

5. A Petri hálók módositásai

a.) A Petri hálók tranziensei *(AND, AND)* tipusuak: minden bemeneti helyről elvesz egy pontot, és minden kimeneti helyre tesz egy pontot. [6]-ban bevezetésre kerülnek *(EOR, AND)*, *(AND, EOR)*, *(EOR,EOR)* jellegü tranziensek. Ha a tranziens input oldala "kizáró vagy"' jellegü, akkor csak egy input helyről vesz el pontot, ha az output oldal *EOR*, akkor csak egy helyre tesz pontot. Az *EOR* jellegü tranziensek használata esetenként kényelmesebb például a döntés *EOR* kimenetü tranzienssel modellezhető. A tranziens gráf változatlanul használható.

b.) Ha egy real time program korrektségével kapcsolatos kérdésekkel probáljuk eldönteni a Petri hálók segitségével, akkor ábrázolnunk kell hálókkal az operációs rendszernek szóló utasitások hatását. Az ütemezési (inditás, inditás periodusokon), kommunikációs (*SEND, RECEIVE*) utasitások hatása még viszonylag egyszerüen leirható. A 6. ábrán látható egy példa. Sajnos a jelenlegi operációs rendszerekben csak ritkán vannak megvalósitva a P és V operációk, és az megnehezeti a vizsgálatokat. A feltétel változók helyett csak flagekat használnak. Például ilyen a *PDP/11 RSX 11/D* operációs rendszere. A flageken a *CLEAR, SET, WAIT* müveletek vannak definiálva. A *CLEAR* és *SET* utasitások mindig végrehajthatók, a *WAIT* utasitás hatására a task feldolgozása megszakad, ha nincs minden flag beállitva. Ezen utasitásoknak a sze-

mantikája a folyamtok az egymáshoz szinkronizálás szempontjából Petri hálókkal leirható (8. ábrán a *CLEAR* és *SET* hatása) de csak bonyolultan, igy elvész a szemléletesség és a gyorsaság. A másik megoldás, hogy a módositott Petri hálóba bevezetjük a flageket is és olyan tipusu tranzienseket is megengedünk mint amilyenek a 7. ábrán láthatók.. Igy ugyan definiciós szempontból a háló bonyolulttá vált és nem lehet könnyen találni hasznos tételeket, de a tranziens gráf módszer továbbra is használható. Ezen pedig a holtpont, szinkronizáció, kölcsönös kizárás vizsgálható. Az eljárások programozhatók, amelyek a vizsgálandó program szövegét inputként kapják. Az értékadásokhoz, predikátumokhoz, szinkronizációs, kommunikációs, ütemezési utasitásokhoz rendelt hálókból a szöveg alapján összerakható a program hálója.

I r o d a l o m

[1] E.W. Dijkstra, Cooperating Sequential Processes. In programming Languages, F. Genuys Ed., Academi Press, New York, 43-112. (1968)

[2] E.W. Dijkstra, Hierarhical Ordering of Sequential Processes, Acta Informatica 1,2 115-115-138 (1971).

[3] C.A. Petri, Concepts of Net Theory, Proceedings of the Symposium on Mth. Foundations of Computer Science, High Tatras, 137-146. (1973)

[4] K. Lautenbach, M.A. Schmid, Use of Petri Nets for Proving Correctness of Concurent Process Systems, Proc. of the IFIP Congress, II. 187-191 (1974)

[5] R.M. Keller, Vector Replacement Systems: A Formalims for Modelling Asyncronous Systems, TR 117, Dept. of EEcs, Prineton Unversity, January, (1974)

[6] J.L. Baer, Modelling for Parallel Computation: A Case Study, Proc. of the Computer Conference on Parallel Processing, Sagamore, (1973)

[7] F.Commoner et all, Marked Directed Graphs, Journal of Computer and System Sciences, 5, 511-523 (1971).

[8] M.H.T. Hack, Analysis of Production Scemata by Petri Nets. MAC TR-94, Massachusetts, 1972.

[9] K. Lautenbach, Exakte Bedingungen der Lebendigkeit für eine Klasse van Netren, Dissertation, Universitat Bonn, (1973)

[10] K.P. Gostelow, Computation Moduls and Petri Nets. TR # 61 Dep. of Information and Computer Sciences, University of California, Irvine, (1965)

[11] K.P. Gostelow, Proper Termination of Flow of Control in Program Involving Concurent Processes, SIGPLAN Notices, 7. 11. nov. (1972).

[12] K.P. Gostelow, A model of Processes Based on Petri Nets, TR # 69, Dep. of Information and Computer Science, University of California, Irvine, (1975)

[13] J.L. Paterson, Modelling of Parallel Systems, Stanford, Ph. D. dissertation, December (1973) (NTIS # PB 231 926)

[14] T. Agarwala, Complete model for Representing the Coordination of Asynchronous Processes, Hopkins Computer Research Report # 32 Computer Science Program, John Hopkins University, Baltimore, Maryland, July (1974).

[15] J.L. Baer, A survey of some Theoretical Aspests of Multiprocessing, Computing Surveys 5, No. 1, March (1973)

[16] R.M. Karp and R.E. Miller, Parallel Program Schemata, J. Computer and System Sciences 3 147-195 (May 1969)

Summary

Graph Modells of Parallel Processes

János Szlankó

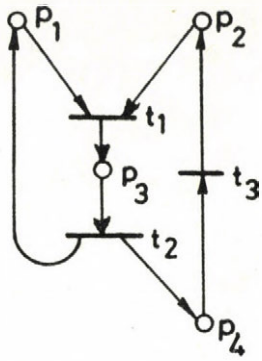
The theory of parallel programming is of great practical importance in the field of operating systems and real time programs. Recently some new graph modells have been introduced. In this area Petri net looks the most promasing formal construction. It has stimulated the researches for further investigations and applications to some practical problems have been demo demonstrated. The article defines the basic concepts of Petri nets and shows the way for mechanisable proofs of parallel program properties.

Р Е З Ю М Е

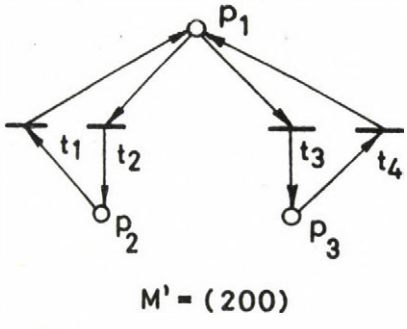
МОДЕЛИ ГРАФОВ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОРОВ

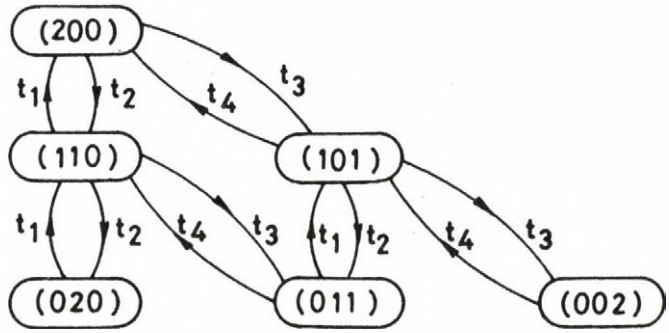
Я. Сланко

Важным практическим вопросом является анализ проведения программ, обрабатываемых параллельно. За последние годы в этой области были сделаны попытки с разными моделями графов. Самой хорошей из них является сеть Петри, которая способствовала возникновению более обобщенных моделей и возникло несколько областей применения. Данная статья рассматривает основы теории сетей Петри и автоматизацию доказательств корректировки.



1. ábra



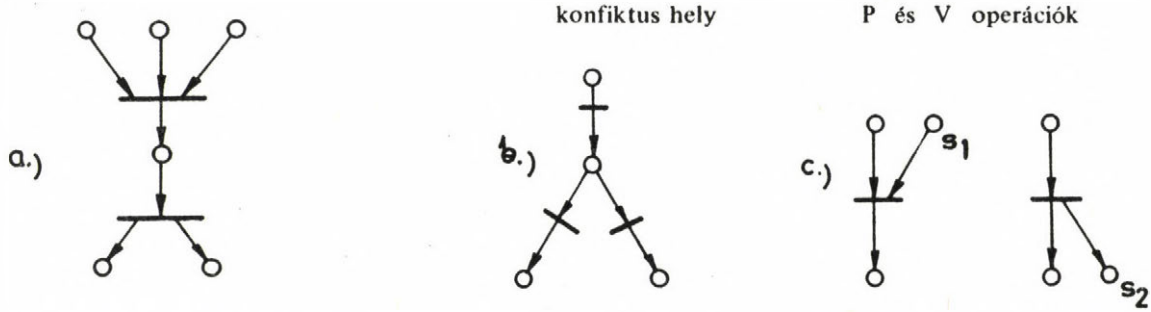


2. ábra

a folyamat indulásához
szükséges feltételek

konfiktus hely

P és V operációk

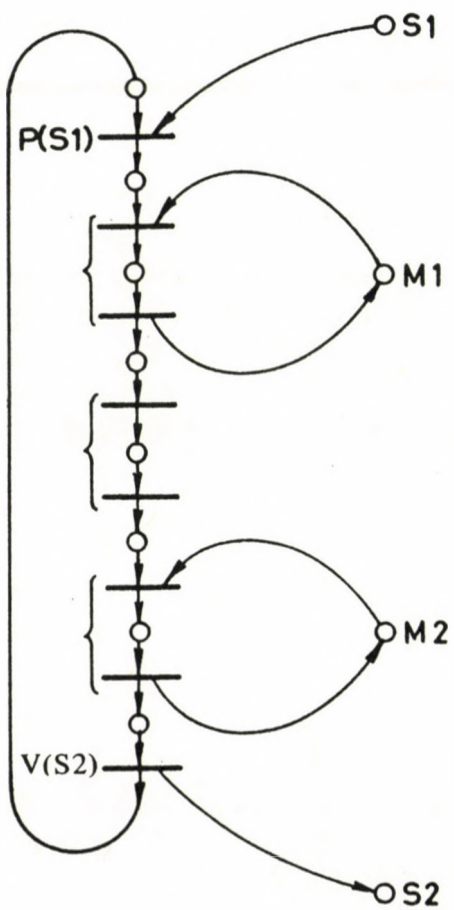


a folyamat befejeződésekor
előálló feltételek

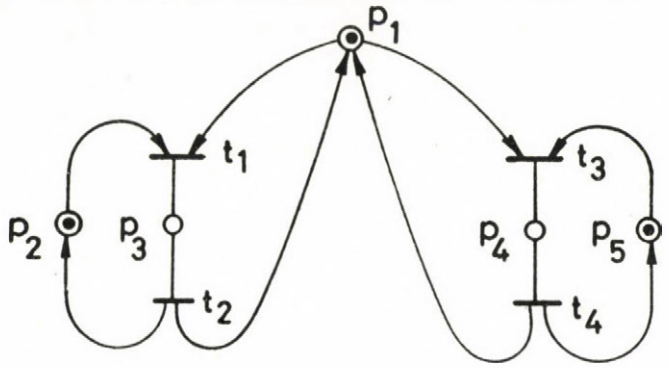
3. ábra

```
C: P(S1);
   P(M1);
   input bufferból olvasás;
   V(M1);
   feldolgozás:
   P(M2);
   output bufferba irás;
   V(M2);
   V(S2);
   goto C;
```

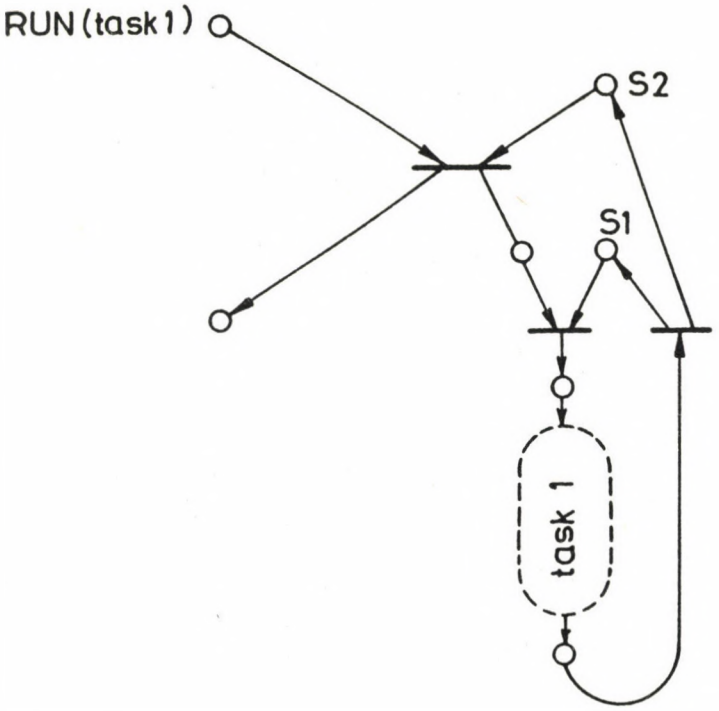
```
P(M1);
input bufferból olvasás;
V(M1);

feldolgozás;

P(M2);
output bufferba irás;
V(M2);
```



4. ábra



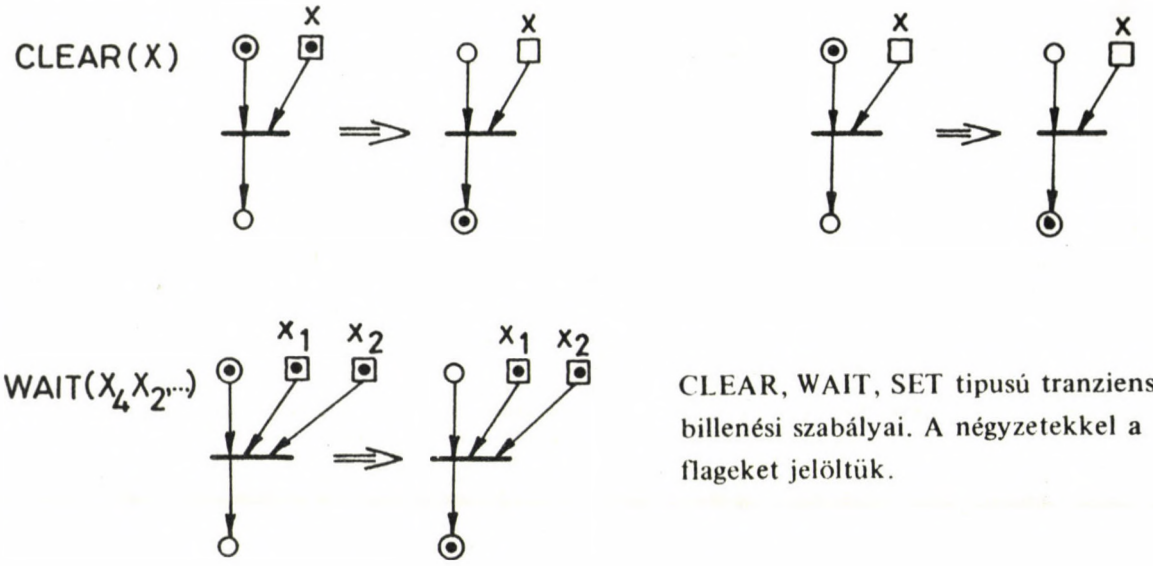
5. ábra



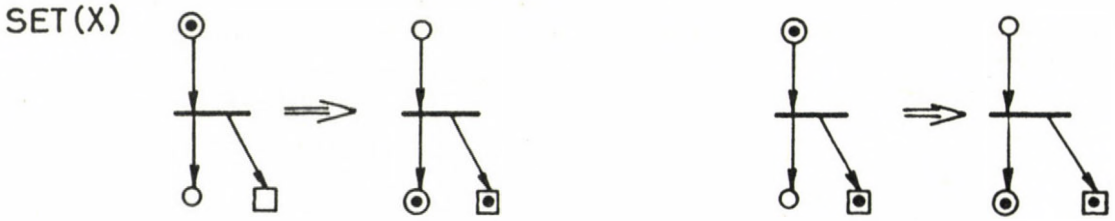
kezdetben: $M(S_1) = 1$

$M(S_2)$ = várólista max hossza

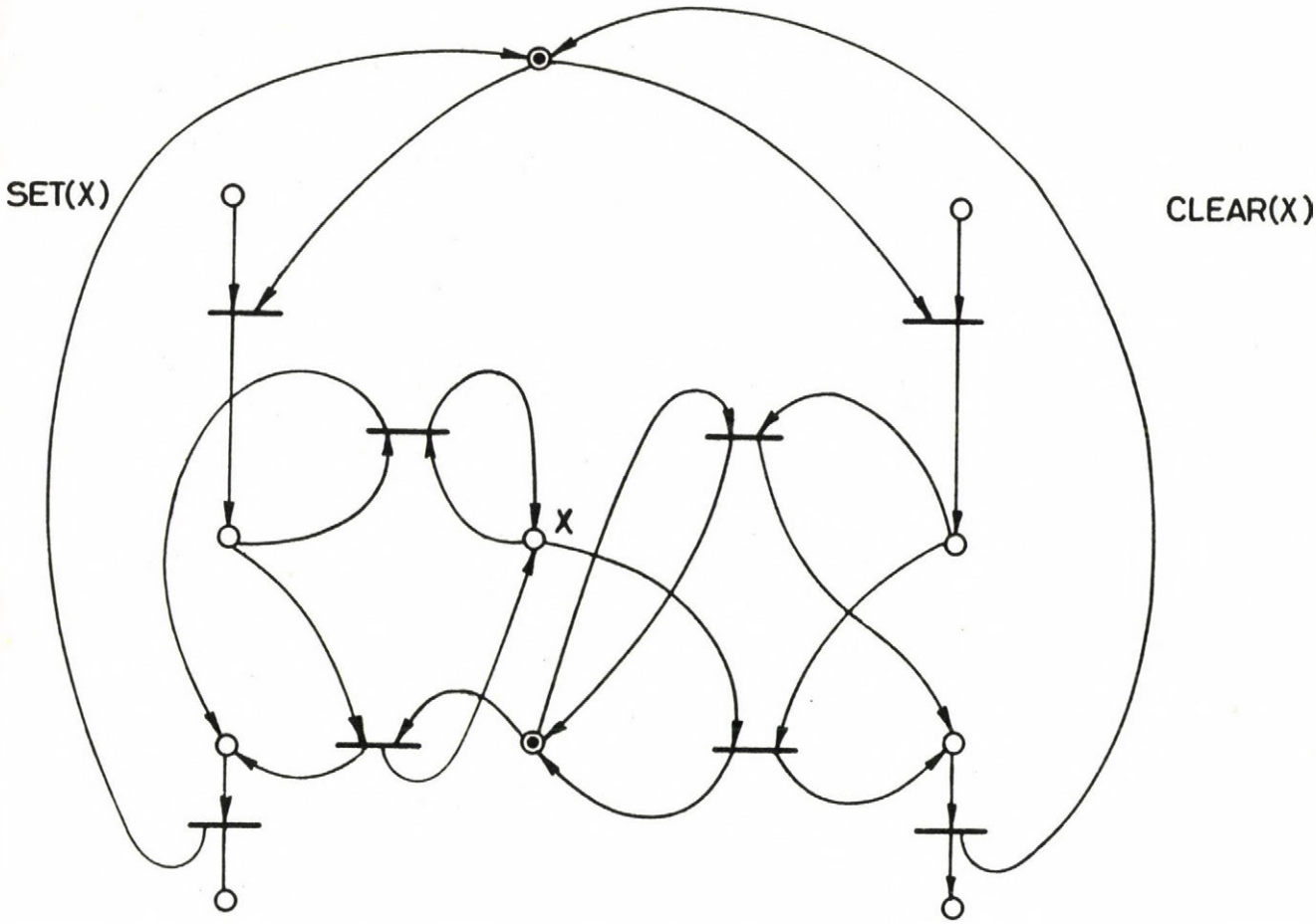
6. ábra



CLEAR, WAIT, SET tipusú tranziensek billenési szabályai. A négyzetekkel a flageket jelöltük.



7. ábra



8. ábra

MTA Számitátechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

REMARKS ON CONCURRENT PROGRAMMING PROBLEMS

Előd Knuth
Computer and Automation Institute of Hungarian Academy of Sciences

I. SYMMETRY CONCEPT IN CONCURRENT ALLOCATION

1. The cyclic allocation problem

We introduce this problem as an extensive generalization of the well-known mutual exclusion problem, see e.g. Dijkstra [1].

Let S be a (finite) set of exclusively accessible resources and P be a (finite) set of cyclic processes. Each of the processes consists of two sections called free section and allocation section. Entering the allocation section needs engaging a definite subset $S_i \subset S$ prescibed by the process $P_i \in P$.

As it is usual, we do not suppose anything of the speed of the processes, so the sequence of events is completely unpredictable.

The problem is to construct suitable algorithms for the processes which satisfy the allocation demands. The difficulty is the same as in the mutual exclusion problem, that is, the execution of any algorithm is time consuming with unpredictable execution time therefore there may well occur conflicts among the demads arisen.

The requirements of the solution we use are comprised in the following points A and B. We remark that these are not the only possible requirements of pratical interest.

A.) SATURATION

A process necessarily must not be suspended if its demands are satisfiable. (This heuristic definition will be exactly reintroduced in part II.)

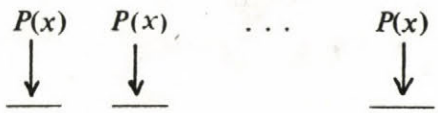
B.) SYMMETRY

Suppose that once resources are released and a set P' of processes of satisfiable demands comes to existance. Now symmetry means that any of the elements of P' has the same possibility to seize its resources (and continue its progress).

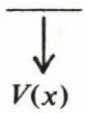
2. On the symmetry concept

As an axiom we suppose the symmetry of the usual semaphore operations. This is illustrated in the following figure.

Suspended processes:



Alarming process:



Now we suppose that $V(x)$ causes an alarm of one and only one waiting process to continue its progress and we have no information about which will really do. All the suspended processes have the same chance.

Remark: It is no matter that the semaphore primitives really have this property or not. However, the requirement "a solution must be symmetric if P and V are" is a proper logical formulation of the fact that we are not allowed to cause any static priority in our algoritms. (To construct priority systems, first we must be able to build symmetric, priority less systems).

3. Parallel semaphore operations

Parallel (multiple) semaphore operations have introduced by several authors, e.g. Patil [2]. The reason of introducing them is the deadlock situation caused by the use of sequential P operations. The effects of the operations are the following:

The execution of the operation:

$$P(x_1, x_2, \ldots x_k)$$

causes suspension if not all the semaphore values seized are positive, otherwise all semaphore values are decreased by one and the executing process continues its progress.

The execution of the operation

$$V(x_1, x_2, \ldots x_k)$$

increases all the semaphore values by one and performs the necessary alarms for the waiting processes.

It is obvious that if these operations may really be put into practice, the cyclic allocation will be no problem, for assigning a semaphore to each resource we could program our processes simply by :

LOOP:

free section;
$P(\cdot\ ,\cdot\ ,\ldots,\cdot)$;
allocation section;
$V(\cdot\ ,\cdot\ ,\ldots,\cdot)$;

4. Definition of parallel operations

Giving an exact definition is far from trivial, and there are several possibilities. Now we quote the version of Patil.

If not all the semaphore values $x_1, \ldots, x_k$ are positive the execution of the operation

$$P(x_1, \ldots, x_k)$$

suspends the executing process placing it into the waiting list corresponding to the first semaphore having nonpositive value. Any later reactivation of the process causes reexecution of the operation which may well result suspension again in a queue corresponding to another semaphore.
The operation

$$V(x_1, \ldots, x_k)$$

must alarm the waiting lists corresponding to semaphores of values having become positive by the execution. However, it is not sufficient to remove one process from the waiting list. All the waiting processes must be alarmed as we have no information of which of them can continue.

No doubt, we could hardly introduce these complicated operations as "primitives". This was correctly pointed out by Parnas [3]. It is noted in the same paper that the parallel operation can be programmed in a "straightforward" way using P and V and conditionals. Let us take a closer look at this question:

5. Implementation problems of parallel operations

Let us denote $P_1 \leftrightarrow P_2$ if the subsets of resources needed by the two processes are not disjoint. Consider the equivalence classes generated by the transitive closure of the relation $\leftrightarrow$, and assign mutual exclusion semaphores to each class. Let m be one of them.

Let $x[\ ]$ be the semephore array representing the resources (initially 0) and $y[\ ]$ be a corresponding integer array having initial values 1.

Replace the parallel P operation by the following sceme:

```
entry: P(m);
    for i: = 1 step 1 until k do
        if y[i] < 1 then
            begin
                V(m);
                P(x[i]);
                goto entry
            end;
    for i: = 1 step 1 until k do
        y[i]: = y[i] − 1;
    V(m);
```

If we can prove this version works correctly the following problem will remain unsolved: How many $V[\ ]$ operations must be performed by our implemetation of parallel V operation. Solving this question the program becomes a little more complicated, but the solution is in fact possible.

However, now I am not concerned to give a complete implementation or discussion of its correctness. The only thing I call attention to is that we are not fully agreed with Parnas on the "straightforward" possibility of implementation. Of course, there are several other possibilities of the implementation but we have not heard about a solution simpler than this, and we would be glad to hear of it if any exists.

Be that as it may, the main inconvenience of these implementations is caused by static priorities which are unavoidable whenever the resources are consistently sequentially tested and released.

6. Ensuring the symmetry

There are famous particular cyclic allocation problems such as the so-called "Dining Philosophers Problem" solved by Dijkstra [4] using only single semaphore operations, and the so-called "Cigarette Smokers Problem" having been considered unsolvable for Patil tried to prove its impossibility while Hebermann [5] and Parnas [3] found a crafty solution for it.

These solutions are built on particular tricks, however, we found an extreme generalization of Hebermann's technique which led to a general existence theorem:

Arbitrary cyclic allocation problem can be programmed symmetrically using only the simple P,V primitives (and even without using conditionals).

This result will be published in the next issue of the Hungarian journal Acta Cybernetica (1976).

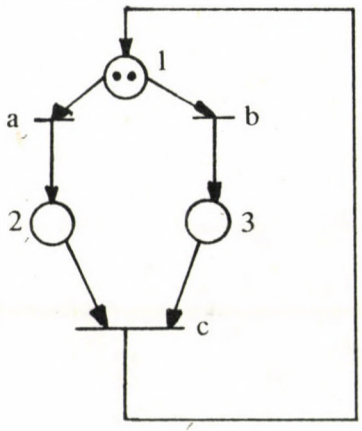
II. A PROBLEM IN THE THEORY OF PETRI NETS

1. Petri Nets

Now I give a very brief introduction to Petri Nets; one can find detailed investigations in the works Holt-Commoner [6] and Petri [7].

A Petri Net is defined to be a labelled directed graph with two node types called places and transitions such that every edge connects a place to a transition or a transition to a place.

A marking M is a function from place labels into nonnegative integers. For example:



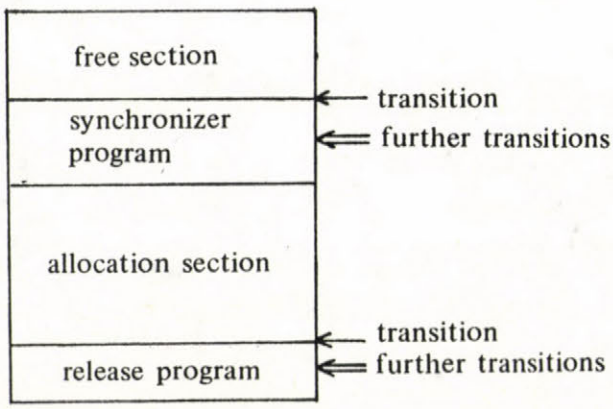
$M = [2,0,0]$

The marking of a net can be changed by firing a transition. A transition is firable if its input places are all positively marked. The firing decreases the number of markers at all the input places by one, and increases the outout markers by one.

Petri Nets can be used to answer a number of interesting questions in connection with concurrent programs for example the deadlock problem.

2. Verification of cyclic allocation systems

Suppose we have a solution and we are intending to verify its correctness. Then each of our cyclic processes has the following form:



Suppose we have made a Petri Net representing our solution. Then we can use the usual methods to prove that the system is free of deadlocks, there may not be conflicting processes in their allocation section at the same time, and other properties.

However, what about the saturation condition?

It is obviously not desirable that a process must really not be suspended if its demands are satisfiable for releasing resourcces and recognizing the free resources surely needs firing a positive number of transitions.

So, we should weaken our condition:

"If the allocation demand is satisfiable then the process must not wait for long."
To describe this exactly we introduce the

3. Classification of transitions

Let T be the set of transitions of a Petri Net. Select a subset $S \subset T$ called significant transitions.

We define the net (regarding an initial marking and a set of significant transitions) to be regular if the net is free of deadlock, but if we prohibit the significant transitions from firing at any time then the system will necessarily get into a deadlock within firing a finite number of transitions.

The states (markings) reached are called main states. A main state is characterized by the property:

If a transition is firable then it is a significant transition.

Reintrodicing the SATURATION condition:

A solution of the cyclic allocation problem satisfies the condition if the Petri Net representing the system with two main transitions: leaving the free and the allocation section; is

a.) regular
b.) if the system is in a main state and the resources used by a process are all free then the transition at which the process rearched the local deadlock may not be after the free section and before the allocation section at the same time.

4. Problem

We propose to study the regularity of the nets with given significant transitions, and produce the possible main states.

A system which is not regular might be considered one having the generalized form of the "infinite after-you blocking" introduced by Dijksta [1].

Further, we remark that it is no matter that all the solutions of the cyclic allocations can be described by Petri Nets or not. Obviously not. We aimed to propose the regularity problem and call attention at the transitions of Petri Nets being able to have different roles (even in the net structure).

Naturally, this problem can be proposed not only for Petri Nets but for more general computational structures too.

References

[1] E.W. Dijkstra, Cooperating Sequential Processes. Programming Languages (1968).

[2] S.S. Patil, Limitations and capabilities of Dijkstra's semaphore primitives for coordination among processes. Proj. MAC, Computational Structures Group Memo 57. (1971).

[3] D.L. Parnas On a solution to the Cigarette Smoker's Problem (without conditional statements). CACM 181-184, (1975).

[4] E.W. Dijkstra , Hierarchical Ordering of Sequential Processes. Operating Systems Technincs (1972).

[5] A.N. Habermann, On a solution and a generalization of the cigarette smoker's problem. Techn. Rep. Carnegie-Mellon University, (1972)

[6] F. Commoner, S. Holt, Marked directed graphs. J.Comp. and System Sci. 5.5. 511-523, (1971).

[7] C.A. Petri, Concepts of net theory, Proceedings of the Symp. on Mathematical Foundations of Computer Science. High Tatras, 137-146. (1973).

Összefoglaló

Konkurens programok konstruálásáról

Knuth Előd

A dolgozat bemutatja az u.n. ciklikus allokáció feladatát, ismerteti a megoldás nehézségeit, és bevezeti a Petri-hálók regularitásának fogalmát, mely fontos szerepet játszik az allokációs rendszerek verefikálásában.

Р Е З Ю М Е

О КОНСТРУКЦИИ КОНКУРРЕНТНЫХ ПРОГРАММ

Е. Кунтх

Работа показывает задачу о так называемом "циклическом распределении ресурсов", трудности решения, и вводит понятия регулярности сетей Петри, которое играет важную роль в верификации систем распределения объектов.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

SCHEDULING SPLITTABLE TASKS ON PARALLEL PROCESSORS TO MINIMIZE SCHEDULE LENGTH

Jacek Błażewicz, Wojciech Cellary, Jan Weglarz
Institute of Control Engineering, Technical University of Poznań,
Poznań, Poland

1. Introduction

In deterministic scheduling problems, knowledge of all task execution times, task arrival times and the precedence relations among tasks is assumed. In practice, these assumptions are rarelly directly met; in particular it is usually not possible to know a priori the task execution times; one would at best only have probability distributions for these times. But the analysis of deterministic problems allows the system designer to determine how well the operational scheduler does in comparison to an optimal one and to provide guigance in improving the scheduler. Moreover, the execution time τ_j of task T_j has at least two practical interpretations [3]. Firstly, it may be the upper bound of the task execution time. Scheduling using these values is equivalent to worst case analysis and is applicable to the "hard-real-time" environment with strict deadlines to be observed. Secondly, τ_j may denote the expected value of the execution time of task T_j considered as a random variable. In this paper the problem of scheduling splittable tasks to minimize schedule lenght is considered. An algorithm for finding the optimal schedule is presented for the case of dependent tasks and identical processors. The idea of solving the proble, for heterogeneous processors is also given.

2. Basic assumptions

It will be assumed that the set of tasks $T_1, T_2, \ldots, T_n$, the set of processors $P_1, P_2, \ldots, P_m$, the structure of precedence relations among tasks, and task execution times τ_{ij}, $i = 1,2, \ldots, m$, $j = 1,2, \ldots, n$, where τ_{ij} denotes the execution time of task T_j on processor P_i, are given.

The structure of precedence relations among tasks is usually given in the form of a precedence graph, where nodes correspond to tasks and arcs to precedence relations. A simple example of a precedence graph is shown in Fig. 1.

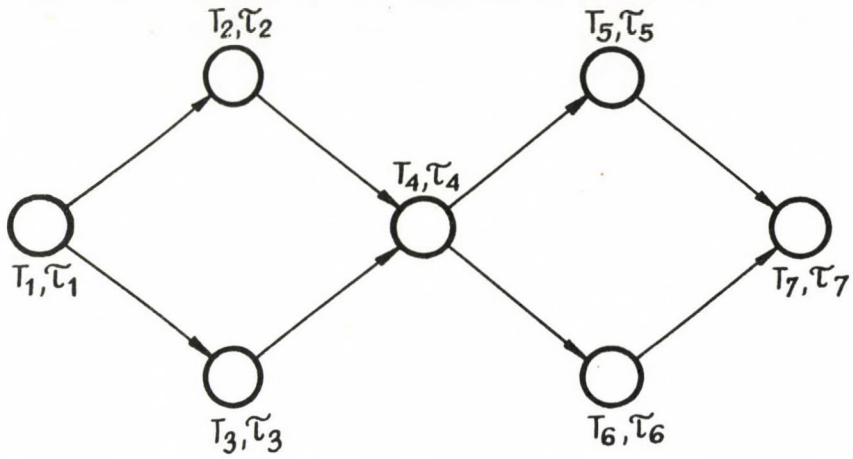


Fig. 1. An example of a precedence graph

Following the most commonly used convection $T_j < T_j$ denotes the fact that task T_i precedes task T_j i.e. task T_j cannot begin processing until task T_i is completed. If no precedence relation among tasks exists, then the tasks are called *independent*, otherwise they are called *dependent* or *precedence related.*

The set of tasks is to be processed on m parallel processors i.e. every processor is capable of processing every task. If the execution times of every individual task are equal one to another for all the processers, then the processors are called *idèntical.* Otherwise, i.e. when there are differences among the processing speeds of the processors, the processsors are called *hęterogeneous.*

A very important feature of a task is whether it may be preempted and then completed, not necessarily on the same processor. Generally, the possibility of task preemption is profitable for some measures for schedule evaluation. If preemption is allowed, it is usually assumed that the processing of the preempted task may resume where it left off without any extra work or time being occasioned by the preemption. This is called a preempt-resume discipline [5], In computer applications, this will not generally be the case, for preemptions may imply the removal of tasks from core storage and subsequent reloading of these tasks. Moreover, in those cases when input/ output operations cannot be overlapped sufficiently, the time delays introduced will be so high that consideration of preemption can frequently be ruled out a priori.

On the other hand examining the preempt-resume discipline is of great importance in systems of parallel processors using a common operating store. Such systems have increasingly many applications in the control of such processes as traffic, telephoneswit control organiza-

tion [6, 12] in which several processors using a common data base computational procedures are being used. It is easy to verify that the possibility of preemption is profitable for improving the schedule lenght.

3. Scheduling splittable tasks to minimize schedule lenght

In this Section scheduling splittable tasks to minimize schedule lenght will be detailed.

Scheduling such tasks has been considered in [9, 10, 11]. The algorithms presented in the these papers concern homogeneous processors and relatively simple precedence relations among tasks. The problem of scheduling independent tasks on processors that are consistently fast or consistently slow for all the tasks was considered in [4, 8]. In the papers mentioned above, non-enumerative algorithms were presented. However, the problem of scheduling dependent, splittable tasks, in the general case, is known to be polynomial complete [4] and hence unlikely to admint a non-enumerative. Thus, for this case, the direct use of scheduling strategies in an oparating system has rather restricted applications. Finding such strategies has, however, practical significance for the following reasons. Firstly, one can use them to estimate an operational scheduler. Secondly, the distance between an optimal solution and a suboptimal one for a heuristic, non-enumerative approach, may be found. Thirdly, enumerative algorithms may be used in computer centres taht perform large and complex numerical computations, but not in a real-time environment.

Below, such scheduling strategies will be presented, which yield some particular advantages [1].

Let us assume, that the processors are identical. Thus, task T_j is characterized by execution time τ_j, $j = 1,2, \ldots, n$. The precedence relations among tasks are given in a form of an activity network (i.e. an acyclic, directed graph with only one origin and only one terminal node) in which arcs correspond to tasks and nodes to events. The precedence graph from Fig. 1. is presented in Fig. 2. in this form.

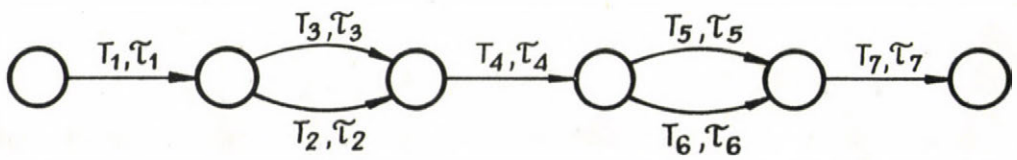


Fig. 2. The set of tasks from Fig. 1. in the form of an activity network

Let the number of nodes of the network be equal to $r + 1$. It is assumed that events are numbered so that event i is not later than event j, when $i < j$. The set of all tasks which can be processed between the occurrence of event k and $k + 1$ will be called the main set and denoted by S_k, $k = 1,2, \ldots, r$. The number of elements of set S_k will be denoted by $|S_k|$.

Now, let us number from 1 to N the feasible sets, i. e. those subsets of all main sets, in which the number of elements is not greated than m.

Let Q_j denote the set of all numbers of the feasible sets in which task T_j may be processed and x_i the duration of set i. Thus the linear programming problem is obtained [1]:

Minimze

$$y = \sum_{i=1}^{N} x_i$$

Subject to

$$(1) \qquad \sum_{i \in Q_j} x_i = \tau_j \qquad j = 1,2, \ldots, n$$

or in matrix notation

$$(1') \qquad \underline{A}\ \underline{x} = \underline{\tau}$$

where $\underline{A}$ is the matrix of coefficients:

$$a_{ij} = \begin{cases} 1 & \text{if} \quad i \in Q_j \\ 0 & \text{otherwise} \end{cases}$$

Obviously, the columns of matrix $\underline{A}$ correspond to the feasible sets. The number of variables in the above problem for m processors is given by the formula:

$$\sum_{k=1}^{r} \left(\sum_{i=1}^{m} \binom{n}{i} k \right) - \sum_{i=1}^{m} \binom{n'}{i} k \Bigg) ,$$

where n_k is the number of elements of the main set S_k;

n_k' is the number of elements of the main set S_k, which composed at least one set $S_1, S_2, \ldots, S_1$, where $1 = k - 1$.

It is assumed in the above formula, that $\binom{n}{i} = 0$ for $n < i$.

It is clear that the number of variables increases rapidly when the number of tasks and processors increases. For example for 5 processors and not very complicated networks containing to task the number of variables is equal to 50, 30 tasks – $2 \cdot 10^3$, 60 tasks – $3 \cdot 10^4$, 100 tasks $2 \cdot 10^5$. If we want to use one of the simplex methods directly for solving the

last problem, 10^7 cells of memory will be needed because of the necessity of memorizing the matrix $\underline{A}$.

As follows from the above, the direct use of simplex methods has a very restriced application.

Bellow, an approach which allows for the great reduction of storage requirements will be shown [1]. On the other hand, in the revised simplex method [7] the elements of the matrix of coefficients (i.e. matrix $\underline{A}$) are not changed during complication. This allows for the generation of single columns of this matrix (or in the other words of feasible subsets of the main set S_k, $k = 1,2, \ldots, r$) in every simplex iteration. Thus, at every moment only one main set and one of its feasible subsets has to be memorized. The use of revised simplex method is also more efficient [7], because in the described problems the number of variables is more than three times greater than the number of constraints.

Now the concept of the generation of feasible subsets will be presented. The number of processors – m, and the *network structure vector* are read as input data. The network structure vector contains the consecutive tasks as the ordered pairs of the nodes. Using the above data, consecutive main sets are generated. First, the tasks which were members of the main sets generated earlier (we will call them old tasks) are added to the main set currently being generated, and then the tasks which composed only this set (we will call them new tasks) are added. Using this information, all feasible and different subsets of all main sets are created. In Fig. 3, the block diagram of the generation of the main set S_k is shown, and in Fig. 4, the block diagram of the generation of the feasible and different subsets of the main set S_k is shown, and in Fig. 4., the block diagram of the generation of the feasible and different subsets of the main set S_k is shown. The point in which the generated subset is used in a simplex iteration is denoted by the block "revised simplex procedure".

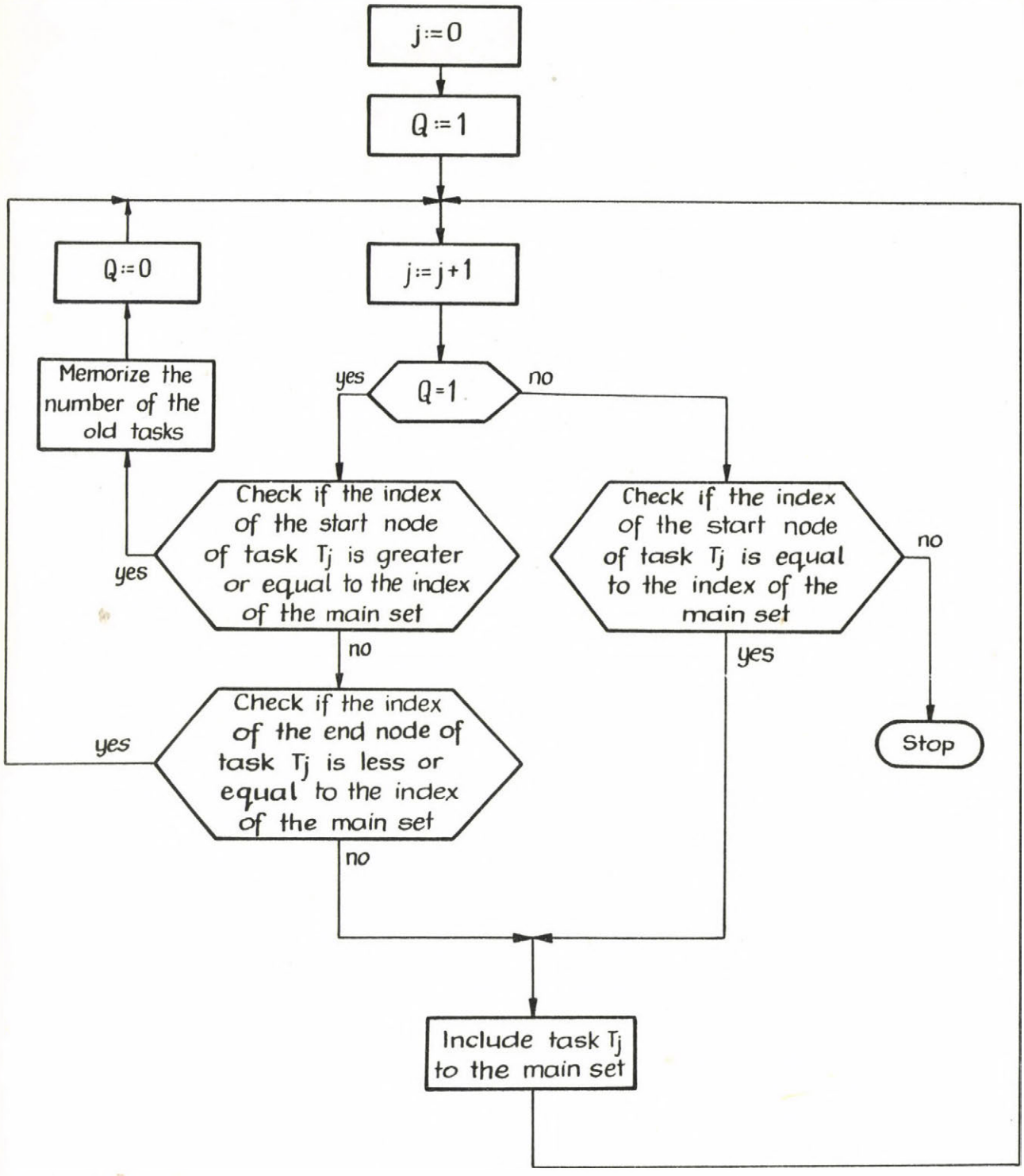


Fig. 3. Block diagram of generation of main set

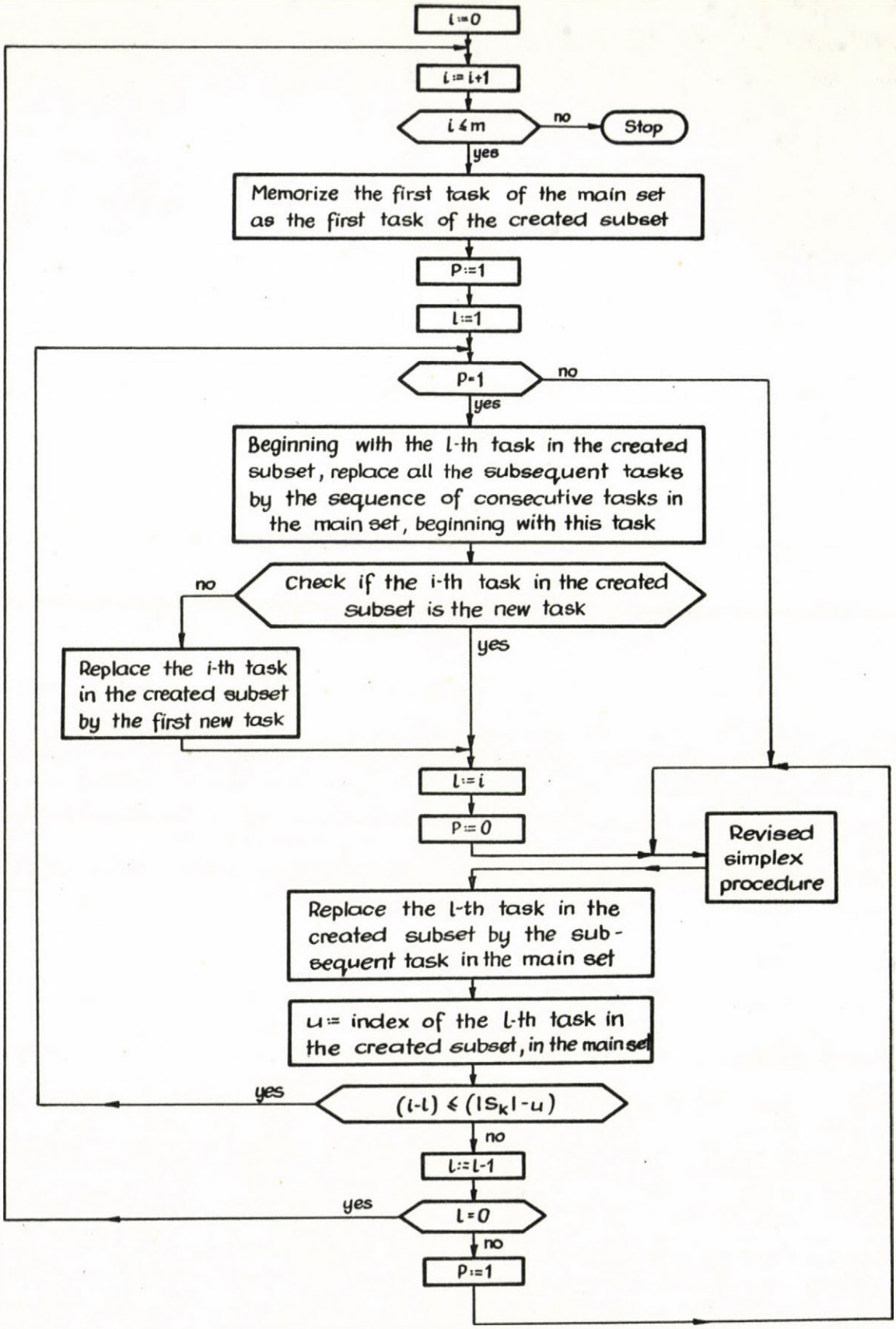


Fig. 4. Block diagram of generation of subsets of main set

Of courge, if the network, node ordering is not given, the obtained schedule is in general a suboptimal one. The optimal schedule may be obtained by choosing the best one from among optimal solutions for all possible orders.

Now let us consider the case of heterogeneous processors [2]. We introduce the following additional denotations:

- K_j, $j = 1,2, \ldots, n$, the set of indices of these main sets in which task T_j may be processed;
- $x_{ijk\epsilon} <0,1>$, $i = 1,2, \ldots, m$; $k\epsilon K_j$, a part of task T_j processed on processor P_i in S_k;
- y_k, $k = 1,2, \ldots, r$, the shedule lenght in S_k;
- $y = \sum_{k=1}^{r} y_k$, the schedule lenght.

Using the above denotations, the following linear programming (LP) problem may be formulated:

Minimize y

Subject to

$$(2) \qquad \sum_{k \epsilon k_j} \sum_{i=1}^{m} x_{ijk} = 1 \qquad j = 1,2, \ldots, n,$$

$$(3) \qquad y_k - \sum_{j \epsilon S_k} x_{ijk} \tau_{ij} \geqslant 0 \qquad \begin{matrix} j = 1,2, \ldots, m \\ k = 1,2, \ldots, r \end{matrix}$$

$$(4) \qquad y_k - \sum_{i=1}^{m} x_{ijk} \tau_{ij} \geqslant 0 \qquad \begin{matrix} j = 1,2, \ldots, n, \\ k \epsilon K_j \end{matrix}$$

Equation (2) guarantees that every task will be processed; inequality (3) defines $y_k's$ as the schedule lenght; inequality (4) assures that it will be possible to obtain a feasible schedule, i.e. one such that no task is processed simulatneously on more than one processor. As the result of solving the described LP probelm, the optimal values y_k^* x_{ijk}^*, $i = 1,2, \ldots, m$; $j = 1,2, \ldots, n$; $k\epsilon K_j$, are obtained. The starting points of parts of tasks x_{ijk}^* are then found [2].

References

[1] J. Blazewicz, Cellary, W. J. Weglarz, Scheduling preemptable tasks on hetegrogeneous processors (submitted for publication).

[2] J. Blazewicz, W. Cellary, J. Weglarz, Some computational problems of scheduling dependent and preemptable tasks to minimize schedule lenght, Foundations of Control Engineering 1, 2(1975).

[3] Jr. E. G. Coffman, P. J. Denning, Operating Systems Theory, Prentice Hall, Englewood Cliffs, N. J. (1973)

[4] Jr. E. G. (ed.) Coffman, Computer and job (shop scheduling theory, Wiley-Interscience (1976)

[5] R.W. Conway, W.L. Maxwell, L.W. Miller, Theory of Scheduling. Addison-Wesley, Reading, Mass. (1967)

[6] A. A. Covo, Analysis of multiprocessor control organizations with partial program memory replication, IEEE Trans. Computers C-23, 2 (1974), 113-120.

[7] S. J. Gass, Linear programming , McGraw-Hill, N.Y., (1969)

[8] E. C. Horváth, R. Sethi, Preemptive schedules for independent tasks, Technical Report No 162, Computer Science Dep., Pennsylvania State Univ. (1975).

[9] Mc Naoghton , R., Scheduling with deadlines and loss functions, Management Sci. 6,1 (1959), 1-12.

[10] R.R. Muntz, Coffman E.G., Jr., Optimal preemtive scheduling on two-processor systems, IEEE Trans. Computers C-18, 11 (1969), 1014-1020.

[11] R.R. Muntz, Jr. E.G. Coffman, Preemtive scheduling of real-time tasks on multiprocessor systems, J. Assoc. Comput. Mach. 17,2 (1970), 324-338.

[12] J. Torres, Test and evaluation of computer traffic control system, vol. 9 of Diamond Interchange Traffic Control, Pb-224160, (1973)

Összefoglaló

Jacek Błażewicz, Wojciech Cellary, Jan Weglarz

Felbontható taskok optimális ütemezése párhuzamos processzorokon

A dolgozat n megszakitható task m processzoron való determinisztikus ütemezésével foglalkozik. A minimális terminálási idő meghatározását lineáris programozási feladatra vezeti vissza. Foglalkozik nem azonos teljesitményű processzorok esetével is.

РЕЗЮМЕ

ПРОБЛЕМА РАСПИСАНИЯ РАЗЛАГАЕМЫХ ПРОГРАММ НА ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОРАХ

Блазениц - Келлари - Велгларз

В работе рассматривается детерминическая проблема n и m процессоров. Задача преобразования в проблему линейного программирования.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

ОБ ИСПОЛЬЗОВАНИИ ВОЗМОЖНОСТЕЙ ОПЕРАЦИОННЫХ СИСТЕМ В ПАКЕТАХ ПРИКЛАДНЫХ ПРОГРАММ

И.Н. Парасюк, И.В. Сергиенко

В настоящее время вопросам создания пакетов прикладных программ (ППП) уделяется много внимания. Различными авторами в этой области прикладной кибернетики предлагаются ключевые понятия и определения, принципы создания ППП, изучаются составные части ППП и их функциональная взаимосвязь, способы машинной реализации ППП и использование возможностей современных операционных систем при их разработке.

Создание ППП на ЭВМ второго поколения сопровождалось преодолением ряда трудностей, связанных, в основном, с ограниченными ресурсами ЭВМ, их быстродействием и наличным математическим обеспечением и другими объективными факторами. В связи с этим в первых ППП приходилось зачастую создавать элементы служб современных операционных систем, таких как загрузчик и редактор связей, управление данными и задачами, управление памятью и др. Разработчиками создавались методы, позволяющие организовать вычислительный процесс в ППП с экономным использованием ресурсов ЭВМ.

Одним из основных принципов создания ППП является развитая модульность его программного обеспечения, позволяющая достигнуть гибкости пакета, т.е. реализации таких принципов

построения ППП как адаптируемость, модифицируемость, расширяемость, совершенствование и др. Кроме того, современные ППП должны быть просты в применении, позволять автоматизировать процессы выбора методов решения прикладных задач и построения соответствующих программ, допускать оперативное сопряжение с другими пакетами, программами, системами и т.п.

В настоящем докладе будем изучать ППП, которые функционируют под управлением операционной системы (ОС) в режиме пакетной обработки данных. Такой ППП в ОС рассматривается как проблемная программа, выполняющая в одном из ее разделов. ОС осуществляет общее управление вычислительной системой, операциями обмена данных, вычислительным процессом в целом, обеспечивая при этом оптимальное использование ресурсов системы.

Процесс решения задач с помощью такого ППП может быть представлен схемой рисунка I.

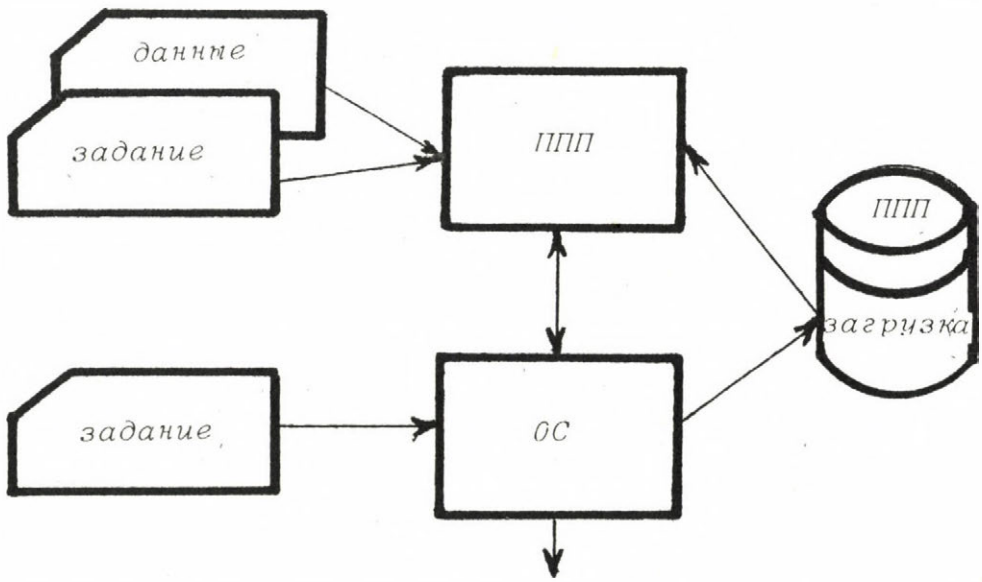


Рис. I

Задание на решение прикладных задач с помощью ППП состоит из двух составных частей: задания для ОС и задания для ППП. В задании для ОС на языке управления заданием [2] описываются ресурсы ЭВМ, необходимые для работы ППП, данные, с которыми ОС должна будет работать, а также сведения о размещении в памяти ЭВМ резидентной части ППП*), которую необходимо загрузить в свободный участок основной памяти и осуществить ее запуск. В задании ППП во входном его языке описывается режим работы ППП, задачи, которые необходимо решать, формат, тип и носитель данных, с которыми будет работать ППП. Задание ОС и задание ППП последовательно включаются во входной поток ОС и обрабатываются соответственно управляющей программой ОС и управляющей программой ППП. Когда управление передано управляющей программе ППП (ее резидентной части, в дальнейшем просто - р е з и д е н т у), последняя согласно заданию пакета с помощью средств ОС и под ее контролем осуществляется построение алгоритма (программы) решения прикладных задач и его выполнение. В некоторых случаях, например, когда некоторые задания ОС систематически используются, их целесообразно оформить в виде каткологизированных процедур. Это значительно упрощает процесс подготовки задания [3].

В современных ППП можно выделить следующие основные части (программы):

---

*) Под резидентной частью ППП понимается та часть управляющей программы ППП, которая в процессе работы ППП постоянно находится в основной памяти. Хранить весь ППП в основной памяти нецелесообразно, так как при этом не рационально используется основная (дорогостоящая) память. Выделение резидентных частей применяется в настоящее время во многих системах ППП и т.п.

I) у п р а в л я ю щ а я программа пакета, которую часто называют монитором пакета или ведущей программой пакета;

2) программа у п р а в л е н и я д а н н ы м и пакета, которую также называют администратором пакета;

3) программа с о в е р ш е н с т в о в а н и я пакета;

4) библиотека баз д а н н ы х пакета, называемая иногда в упрощенных вариантах ППП библиотекой пакета, телом пакета;

Программа управления данными совместно с библиотекой баз данных составляют б а н к д а н н ы х ППП.

Функциональная взаимосвязь этих программ ППП схематически показана на рис. 2.

На этом рисунке стрелочки "⇒" указывают информационные связи, а стрелочки "→" - связи по управлению.

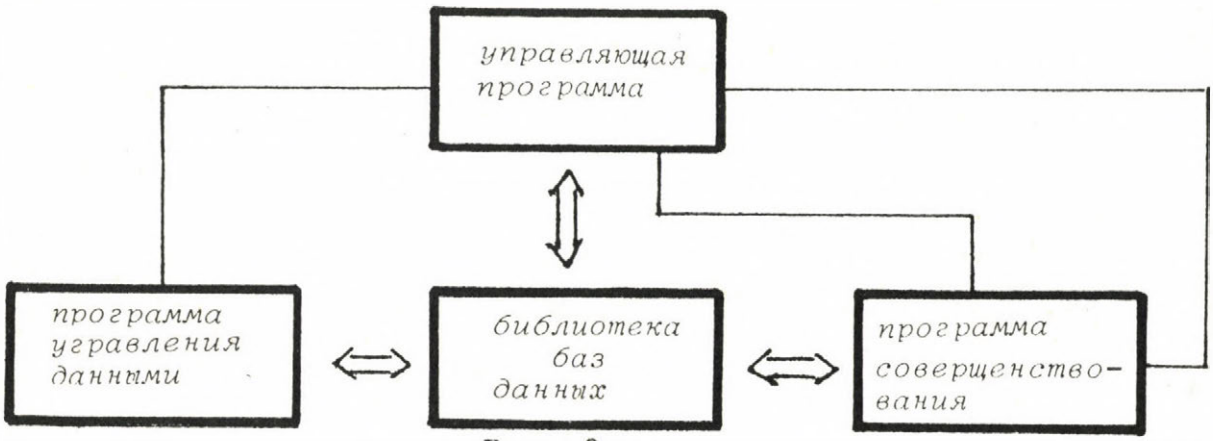


Рис. 2

Для работы ППП целесообразно предусмотреть следующие режимы.

I. Р е ж и м с ч е т а. Это основной режим работы ППП. В этом режиме осуществляется выбор необходимых математичес-

ких методов для решения заданной совокупности проблемных задач и организация процесса их решения. Режим счета применяется в ППП как автоматно, так и в сопряжении с другими программами.

2. Р е ж и м к о н с т р у и р о в а н и я. В этом режиме осуществляется конструирование ППП программ, записанных на исходных языках высокого уровня типа АЛГОЛ, ФОРТРАН, PL/1 [2] и др. и предназначенных для решения определенной совокупности проблемных задач и организация их записи на заданные носители информации (ПК, ПФ, МД, АЦПУ и т.п.).

3. Р е ж и м с о в е р ш е н с т в о в а н и я. Этот режим предназначен для програмной реализации идей совершенствования математического обеспечения ППП, основанного на накоплении и обработке "опыта" его функционирования [I].

4. Р е ж и м у п р а в л е н и я. Этот режим условно объединяет все операции по обновлению, расширению, отладке, выдаче копий математического обеспечения ППП и различного рода справок о нем (о его использовании, возможностях, хранении, инструкциях и т.п.).

Об особенностях и способах реализации этих режимов в ППП мы раскажем несколько ниже. Сейчас рассмотрим вкратце другой круг вопросов, связанных с созданием ППП, а именно, языки входных сообщений ППП и требования к ним.

Анализ перечисленных выше режимов работы ППП, требований пользователей и особенностей организации вычислительного процесса позволяют сделать вывод о целесообразности выделения в современных ППП двух типов языков, обладающих специфическими особенностями. Прежде всего выделим п р о б л е м -

но-ориентированный язык ППП, как правило, язык очень высокого уровня, предназначенный для описания решаемой совокупности проблемных задач. Этот язык часо называют языком пользователя. Он должен быть сравнительно прост по своей структуре, легок в освоении и применении. Он должен позволять пользователям, как правило, не имеющим специальной подготовки по программированию, формулировать решаемые ими задачи не вдаваясь в логику машинного алгоритма их решения.

Существуют два подхода к созданию проблемно-ориентированных языков ППП. Один из них состоит в расширении языков высокого уровня типа ФОРТРАН, АЛГОЛ, PL/1 и т.п. средствами, ориентированными на решение проблемных задач специальных классов. Процесс формулировки (описания) задач с помощью таких языков, обычно, заключается в том, что составляется проблемная программа, поочередно использующая необходимые средства соответствующего языка высокого уровня и его расширения.

Другой подход к созданию проблемно-ориентированных языков ППП, на котором по ряду причин мы остановим свое внимание, состоит в специальной разработке самостоятельного языка, называемого языком управления заданием ППП. Языки высокого уровня типа АЛГОЛ, ФОРТРАН и т.п. используются в этом случае для написания программных модулей, реализующих определенные фрагменты применяемых алгоритмов. Эти программные модули в различном виде (исходные модули, объективные модули, модули загрузки) совместно с логикой алгоритмов решения задач скрыты от пользователя в ППП. Формулировка задач, подлежащих решению, осуществляется те-

перь лишь с помощью проблемно-ориентированных языков ППП очень высокого уровня. Она включает описание входных данных, результатов, укрупненной модели алгоритма решения задач, режима выполнения этого задания пакетом и другой необходимой управляющей информации. Обычно проблемно-ориентированные языки ППП имеют операторную структуру. Среди них можно выделить управляющие, вычислительные, информирующие, описательные, обслуживающие и др. операторы. Например, в проблемно-ориентированном языке пакета программ математической статистики (первая версия) наряду с операторами начала и конца задания включены операторы, описывающие формат и носители информации для выдачи результатов, ввода исходных данных, а также операторы, позволяющие формулировать задачи статистической обработки данных. Например, операторы оценки статистических характеристик, имитации данных, распределенных по заданному закону, проверки различных гипотез, фильтрации выборки, построения моделей и многие другие, позволяющие решать задачи из различных разделов математической статистики. Выделяют [4] три наиболее распространенных формата операторов: формат операторов языка управления заданиями операционной системы, формат макрокоманд и формат операторов обращения к процедурам. Каждый из форматов содержит поле метки оператора, поле имени оператора и полей параметров и операндов.

В процессе создания проблемно-ориентированного языка ППП следует учитывать принципы расширяемости, адаптируемости и модифицируемости. Это значит, что языковые средства пользователя должны быть открыты для расширения и модификации, а их структура должна позволять без особых затрат переориен-

тировать этот язык на решение задач из другой области. Отсюда вытекают два возможных подхода, которые можно применять при реализации входного языка ППП: I) создать ядро языка, допускающее наращивание новыми элементами и адаптацию к новым областям применения; 2) создать мета-язык, позволяющий описывать языки, ориентированные на новые проблемы.

Важным моментом в реализации языкового обеспечения ППП является создание так называемого в н у т р е н н е г о я з ы к а, общего для всей совокупности языков данного пакета, как, например, в системе УСОД, что позволяет более эффективно адаптировать пакеты на новые области применения.

Второй тип языков предназначен для описания заданий администратору данных и программе совершенствования. Характерной особенностью этих языков является то, что они, как правило, при сохранении структур данных пакета, инвариантны к областям его применения. Опыт показывает, что язык администратора можно построить на базе следующих основных операторов: ВЫДАТЬ, КОПИРОВАТЬ, СКОНСТРУИРОВАТЬ, ИЗМЕНИТЬ, ЗАМЕНИТЬ, УДАЛИТЬ (ИСКЛЮЧИТЬ), УПЛОТНИТЬ, УПОРЯДОЧИНИТЬ, КОНСТРУИРОВАТЬ, РАСПЕЧАТАТЬ, СЛИТЬ, ЧЛЕНИТЬ, ВОССТАНОВИТЬ, ВКЛЮЧИТЬ (ПОПОЛНИТЬ) и др.

В язык программы совершенствования пакета можно включить такие операторы как: ОЦЕНИТЬ, ПОДГОТОВИТЬ, ПРЕДЛОЖИТЬ и т.п.

Рассмотрим вкратце функциональную структуру пакета и назначение ее составных частей.

Управляющая программа представляет собой совокупность управляющих и обрабатывающих программ, которые предназначены для анализа входных сообщений, настройки пакета на определен-

ный режим работы и управления процессом выполнения задания. Управляющая программа обычно имеет динамическую структуру. Она состоит из программы трансляции текста, записанного во входном языке ППП и представляющего задание пользователя пакета, в текст, записанный на промежуточном языке ППП программы анализа этого текста, программы загрузки в рабочие области памяти соответствующих программ, таких как: программа автоматического построения обобщенной вычислительной схемы алгоритма решения задач; программа компоновки программ из исходных модулей или модулей загрузки; программа организации интерфейсов и др.

Рассмотрим основные элементы баз данных ППП.

Под п р о г р а м м н ы м и м о д у л я м и (ПМ) ППП будем понимать автономные, упорядоченные фрагменты алгоритмов решения прикладных задач, описанные на одном из языков высокого уровня, оформленные в виде процедур.

Каждый ПМ имеет свой п а с п о р т , в котором содержатся все данные, необходимые для применения данного ПМ в различных ППП, если это целесообразно. Например, ПМ пакета программ математической статистики (ПМС), разработанного в Институте кибернетики АН УССР и функционируемого под управлением ЕС ЭВМ, напсаны на языке ФОРТРАН, а их паспорта содержат следующие данные: имя ПМ, количество глобальных (внешних) переменных ПМ, параметры глобальных переменных (формат, размерность, относительные адреса в поле глобальных переменных) и др.

П М з а д а ч и - это собранная из необходимых ПМ программа для решения данной задачи. Содержать ПМ задачи це-

лесообразно в тех случаях, когда вероятность того, что данная задача будет включена в задание, сравнительно велика. Определение таких задач - одна из функций программы совершенствования ППП.

Остановимся вкратце на вопросах построения с и с т е м ы п р о г р а м м н ы х м о д у л е й 5 представляющей структуру класса задач. Опыт показывает, что от того, какой выбран метод определения совокупности программных модулей, существенно зависят показатели, характеризующие качество функционирования пакета.

Общий случай, т.е. случай, когда класс решаемых задач, а следовательно, и алгоритмы их решения окончательно не определены (хотя бы на этапе проектирования) связан с многими трудностями при решении вопроса о создании оптимальной в некотором смысле системы программных модулей.

Для случая, когда класс задач, допустимых для решения пакетом, окончательно определен, причем этот класс конечный и такой, что каждой задаче можно поставить в соответствие некий вполне определенный алгоритм ее решения, авторами разработан формальный метод, названный ими " Ψ-метод", позволяющий построить систему программных модулей, оптимальную в том смысле, что число входящих в нее ПМ минимально при следующем ограничении - суммарное количество функциональных операторов [5] при решении всевозможных комбинаций задач - минимальное.

Систему ПМ задач в данном случае можно характеризовать модульным графом [5], т.е. информационным ярусно-параллельным графом, вершинами которого являются программные модули, а информационные дуги указывают на глобальные перемены, вос-

принимаемые и вырабатываемые программными модулями в качестве исходных данных и результатов соответственно. Порядок выполнения программных модулей в этой системе определяется следующими правилами: ПМ одного яруса выполняются в произвольном порядке, ПМ i-го яруса можно выполнить только после выполнения смежных с ним ПМ (i-1),(i-2) ,,,,,,, 0 ярусов.

В ы ч и с л и т е л ь н а я с х е м а алгоритма решения произвольной задачи, т.е. последовательность (цепочка) имен ПМ, выполнение которых приводит к решению данной задачи, на этом графе определяется объединением всевозможных путей, ведущих из заданной начальной вершины (ПМ формата (типа) данных) в вершину, определяющую данную задачу. Система ПМ в пакете ПМС как и в системе УСОД представляется вычислительными схемами алгоритмов отдельных задач, полем глобальных переменных, таблицей паспортов ПМ. Вычислительные схемы алгоритмов хранятся в таблицах, называемые таблицами вычислительных схем алгоритмов, в которых содержатся следующие данные: имя задачи, количество ПМ, составляющих вычислительную схему и сама вычислительная схема.

Таблица вычислительных схем и таблица паспортов ПМ в пакете ПМС реализованы в виде невыполнимых многосекционных подпрограмм языка Ассемблер. Каждая такая секция описывает паспорт ПМ или вычислительную схему алгоритма.

Рассмотрим теперь вопросы функционирования управляющей программы пакета на примере монитора пакета ПМС. Процесс функционирования монитора начинается с загрузки и инициирования его резидента, осуществляющего дальнейшее управление пакетом. Он выполняет функции, инициирования программ и функ-

ции внутреннего управления. Интерфейс между программами монитора и его резидентом осуществляется через область связи путем передачи соответствующих кодов возврата и состояния. Область связи пакета ПМС, являющаяся аналогом поля фиксаторов системы УСОД, загружается резидентом в ОЗУ и используется для двухсторонней связи между программами монитора. В эту область помещается информация, которая используется для внутреннего управления (режим работы памяти, формат данных) адреса некоторых программ и т.п.). Структура области связи в пакете ПМС определяется специально введенной невыполнимой макрокомандой языка Ассемблер, макрорасширение которой представляет собой фиктивную секцию программы, что позволяет при написании программ на языке Ассемблер обращаться к ячейкам области связи по именам.

После загрузки резидентом монитора в оперативную память области связи, загружается в рабочую область программа анализа входных сообщений, которая тут же получает управление. Эта программа один за другим вводит и обрабатывает операторы задания (оператор задания, оператор режима работы пакета, оператор шагов задания). Результаты обработки этих операторов загружаются в область связи. В процессе обработки осуществляется синтаксический и семантический контроль предложений задания. В случае, если ошибка обнаружена хотя бы в одном из предложений шага задания, этот шаг задания аннулируется, а на АЦПУ выдается диагностическое сообщение об ошибках.

В том случае, если ошибок в предложениях шага задания не обнаружено, осуществляется перевод его предложений на внутренний язык пакета и из этой информации формируется рабо-

чий файл, который записывает на МЛ и содержит следующую информацию: список имен задач шага задания, имя формата данных и их размерность.

Процесс анализа и обработки предложений задания сопровождается диагностическими сообщениями на АЦПУ. После обработки последнего шага задания формируется код возврата резиденту монитора.

Следующей за программой анализа входных сообщений резидент монитора загружает программу планирования вычислительного процесса и организации его выполнения. Эта программа выполняет следующие функции: подготавливает к работе систему ПМ и информационные таблицы, необходимые для организации пошагового решения задач, конструирование вычислительной схемы обобщенного алгоритма для решения задач данного шага задания, распределение и контроль оперативной памяти, необходимой для исходных данных и глобальных переменных, подготовка списка фактических параметров для ПМ, загрузка в оперативную память ПМ и организация их выполнения.

Рассмотрим более подробно функциональную схему этой программы.

Выделяются рабочие буферы для набора (списка) задач k-го шага задания, таблицы вычислительных схем алгоритмов данного класса задач, вычислительной схемы обобщенного алгоритма решения задач k-го шага задания и заявка на требуемый объем оперативной памяти поступает к резиденту монитора. После выделения требуемой памяти, загружается в рабочие области список задач k-го шага задания: таблица вычислительных схем алгоритмов и по аналогии с системой УСОД констуиру-

ется (синтезируется) вычислительная схема обобщенного алгоритма совместного решения всей совокупности задач k-го шага задания. Далее, на поле таблицы вычислительных схем алгоритмов загружается таблица паспортов ПМ и определяются адреса паспортов ПМ, входящих в вычислительную схему обобщенного алгоритма. Эти адреса последовательно заносятся в подготовительный список параметров (аналог поля вычислительных схем в системе УСОД).

В процессе анализа паспортов ПМ эта программа осуществляет контроль описания исходных данных, определяет необходимый объем оперативной памяти, который и запрашивается у резидента. Если установлено, что требуемая память не может быть выделена или имеется несоответствие в описании данных, то рассматриваемый шаг задания анулируется. В случае, если никаких аварийных ситуаций не возникло, осуществляется распределение памяти для входных и выходных параметров ПМ в поле глобальных переменных. Для простых глобальных переменных в пакете ПМС память не выделяется (ее значение находится в области связи, а адрес соответствующей ячейки поля глобальных переменных включается в список параметров). Если глобальная переменная - массив и если ей еще не выделено место в поле глобальных переменных, то определяется объем требуемой памяти и посылается соответствующий запрос резиденту. В случае выделения требуемого объема памяти, адрес области помещается в список, параметров и в соответствующую ячейку поля глобальных переменных.

Когда список всех параметров ПМ подготовлен, ПМ загружается в оперативную память, пересылается этому ПМ адрес

списка параметров и передается ему управление. ПМ загружается на одно и тоже поле в ОЗУ. При возникновении аварийных ситуаций управление предлагается программе обработки аварийных ситуаций. Нормальная (без аварийных остановок) работа ПМ завершается возвратом управления системе для подготовки и загрузки очередного ПМ.

Выполнение шага задания влечет аннулирование выделенной для него оперативной памяти и передачу управления на анализ следующего шага задания.

Выше, на примере пакета ПМС был рассмотрен процесс планирования и организации вычислительного процесса в обычном аспекте. Однако, как показывает опыт разработки и применения ряда ППП, что целесообразно к блокам ППП подключить программные средства сбора и обработки статистических данных о функционировании пакета. Обработка, накопленного таким образом "опыта" позволит совершенствовать организацию ППП. Например, уточнить класс решаемых задач, оптимально (в смысле минимизации времени загрузки ПМ в рабочие области) разместить ПМ, их паспорта и необходимые таблицы на внешних носителях информации, оценить уровень агрегации ПМ и задач, и если потребуется, предложить списки ПМ и задач, подлежащие укрупнению, членению, оценить параметры, характеризующие эффективность пакета и др. Для этой цели в определенных точках организующей программы необходимо подключить программы фиксирующие временные и частотные характеристики работы ее составных частей, как это сделано в системе УСОД.

В этой системе накапливаются такие данные как частота использования класса задач, различных форматов (типов) дан-

ных, объем обрабатываемой статистики, частота решения различных задач данного класса, частота использования программных модулей в процессе решения, длина сконструированной программы. Более того, в системе УСОД накапливается статистика об использовании программных модулей и параметры сконструированных программ для случая, если бы процесс конструирования рабочих программ не сопровождался их оптимизацией.

Очевидно, что наличие программ сбора статистики несколько замедляет процесс решения задач, поэтому целесообразно, чтобы была возможность отключать работу этих программ в экстренных ситуациях.

Если пакету передано задание для совершенствования, то подключаются программы, обрабатывающие накопленный "опыт" работы и вырабатывающие проект изменения пакета, который наряду с различными оценками некоторых параметров немедленно сообщается "хозяину" пакета. В случае, если принято решение совершенствовать пакет, то соответствующий проект передается администратору данных пакета, который и осуществляет перестройку пакета.

Идея сопряжения ППП с другими программами в своей основе сходна с использованием ППП в режиме счета, что и позволило объединить их в один режим работы - режим счета. Особенностью организации сопряжения программ является то, что потребителям, вырабатывающим данные для обработки, формирующим соответствующие задачи и потребляющие результаты решения является некоторая активная программа (программа-пользователь). Ею может быть программа, пакет или система. В предположении, что программа пользователь (¶ -программа) (так бу-

дем называть программно-организованного пользователя пакета) обучена формулировать задание ППП и его запуск, ППП должен быть способен понять это задание, реализовать его и,осуществив передачу результатов на заданные поля, возвратить управление ¶-программе (в случае успешной реализации задания).

В работе 6 уже рассмотрены особенности организации сопряжения автоматизированной системы обработки данных и системы моделирования и приведена принципиальная схема сопряжения двух реально действующих систем - системы УСОД, предназначенной для обработки статистических данных и системы СЛЭНГ, предназначенной для моделирования систем с дискретными событиями.

Здесь мы укажем лишь на те принципиальные трудности, которые возникают в процессе разработки и реализации сопряжения ППП с ¶-программами.

I. Организация памяти. Так как на память ¶-программы никаких ограничений не накладывается (обычно это довольно сложные и громоздкие системы), то следует разработать некий универсальный механизм распределения и защиты памяти, который позволял бы поочередно для ППП и ¶-программы использовать выделенную основную память.

2. Организация взаимно-оперативной информационной связи. Общий случай обмена данными между ППП и ¶-программой на одном из шагов их совместной работы заключается в оперативной передаче данных ¶-программой (результатов работы ¶-программы) ППП и наоборот - передача результатов работы ППП ¶-программе для дальнейшей их обработки.

3. Планирование оперативного управления вычислительным

процессом в ¶-программе и ППП с учетом возможных аварийных ситуаций.

Другой аспект применения возможностей современных ППП заключается в автоматизации процесса конструирования программ из ПМ, написанных на исходных языках высокого уровня. Результатом работы ППП в данном случае является исходная программа, записанная на заданном носителе данных. Для работы ППП в этом направлении используется тот же, описанный выше, механизм анализа входных сообщений и построения вычислительной схемы алгоритма. Принципиально новым здесь будет механизм функциональной связи ПМ.

Л И Т Е Р А Т У Р А

I. И.Н. Парасюк, И.В. Сергиенко, Н.И. Тукалевская. Универсально-специализированная автоматизированная система обработки данных на ЭВМ (система УСОД), ж.УСиМ, №2, К., 1974.

2. К. Джермейн. Программирование на IBM/360, Изд-во "МИР", М., 1973.

3. Дж. Донован. Системное программирование, Изд-во "МИР", М., 1975.

4. А.С. Стукало. Вопросы разработки языков управления заданиями в пакетах прикладных программ, В сб. "Вопросы оптисизации и организации вычислений". Изд. РДЭНТП, Киев, 1975.

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

6. И.Н. Парасюк, М.А. Сахнюк. Программный комплекс СЛЕНГ-УСОД, ж. "УСиМ", №2, К., 1974.

Összefoglaló

I.N. Paraszuk, I.V. Szegrienko

Operációs rendszerek tulajdonságainak alkalmazása programcsomagok készitésénél

A cikk az alkalmazott programcsomagok felépitésével, tervezésével foglalkozik. Az általános probléma felvetések és megoldások illusztrációjaként a szerzők konkrét példákat mutatnak be. A programcsomagok belső szerkezetén kivül tárgyalják a programcsomagok illesztésénél, ill. programcsomag is felhasználói program illesztésénél felmerülő problémákat is.

Summary

I.N. Paraszuk, I.V. Szegrienko

Application of operating system's properties in constructing program packages

This work deals with the structure and planning of the application program packages. Some concrete comles are given that illustrate the general problems and their solutions. In addition to the describing of the internal structure of the program packages the outhors presents short discussion on the problems of connecting program packages with cach other or with user programmes

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

TÁROLÓVAL VALÓ GAZDÁLKODÁS KISSZÁMOLÓGÉPEK OPERÁCIÓS RENDSZERÉBEN

Salamon Márton
MTA Központi Fizikai Kutató Intézete

A TPA 70 kisszámológép diszkes operációs rendszerének tervezése során kerestük az operációs rendszerek elméletének e feladatnál alkalmazható eredményeit. Kisszámológépeknél az általánosan jelentkező problémák egy része különös jelentőséget kap a gép viszonylag kis operativ memóriája miatt. A tervezési szempontok között egyik legfontosabb az volt, hogy a rendszer központi része, a Supervisor tegye lehetővé a rendelkezésre álló operativ memória maximális kihasználását a programok számára. A Supervisornak egyrészt ki kell elégiteni a rendszerben futó forditóprogramok (Assembler, Fortran, Basic) igényeit, másrészt lényeges, hogy célrendszerekben is alkalmazható legyen, azaz I/O rendszerét könnyen lehessen bőviteni.

Az operációs rendszer tervezésekor figyelembe kellett vennünk a hardware környezetet:

- A TPA 70 egy 16 bit szóhosszuságu, korszerü kisszámológép minimum 8K szó, maximum 32K szó memória kapacitással.
- Virtuális tároló kezelést vagy overlay szervezést támogató hardware segitség nincs.
- Az operációs rendszernek mind mozgófejes, mind fixfejes diszkkel hatékonyan kell müködnie.
- Előre nem ismert perifériák vezérlésének megoldhatóságát biztositani kell.

Az operációs rendszer vezérlő része a Supervisor. A Supervisor moduláris felépitésü. Különböző (programból kiadott vagy operátori) igények hatására más-más modul vagy modulok müködnek.

Martin az operációs rendszerek tervezéséről irt könyvében 9 tényezőt sorol fel, amelyek szük keresztmetszetet jelenthetnek egy operációs rendszer müködése során. Ezek a következők:

- Memória terület
- Központi egység idő
- Háttér tároló kapacitás
- Csatorna kihasználás
- Mágneslemezes háttértárolóknál a fejmozgatások ideje
- Adatátviteli buffer terület
- Adatátviteli vonalak kihasználása
- Terminálok kihasználsa
- Operátori beavatkozások

Az alábbiakban a memória terület kapacitást vizsgáljuk abból a szempontból, hogy a felhasználói program számára hogyan lehet minél nagyobb területet biztositani. Nem foglalkozunk azzal a kérdéssel, hogy a felhasználó hogyan tudja kihasználni a legjobban a rendelkezésre álló területet.

Kézenfekvőnek látszik, hogy a Supervisor rutinjai (moduljai) közül minél többet a háttértárolón tartsunk, és csak szükség esetén hozzuk be a memóriába. Ez az eljárás azonban

- Központi egység időt
- Háttér tároló kapacitást
- Csatorna időt
- Fejmozgatást

igényel. Látható, hogy egy rutin behozása a memóriába négy másik tényezőnél játszik szerepet, tehát nem mindig ez a takarékoskodás legkifizetődőbb módja, hiszen a rendszer hatékonysága nagymértékben csökkenhet. Éppen ezért nagyon alapos megfontolást igényel az, hogy a Supervisor rutinjait hogyan kezeljük.

A TPA 70 operációs rendszerénél a Supervisor rutinjait három csoportba osztottuk.

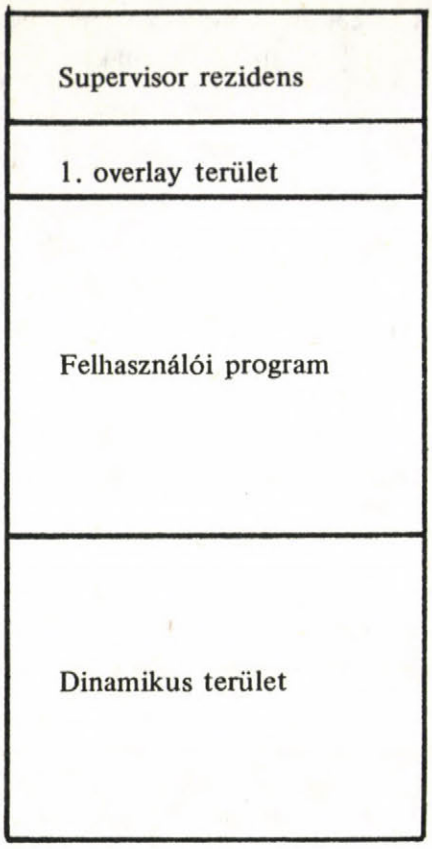
- Rezidens rutinok, amelyekre gyakran, vagy nagyon gyorsan van szükség, ezért állandóan a memóriában vannak.
- Overlay rutinok, amelyek a ritkábban előforduló igények esetén kerülnek be a memóriába.
- Perifériák handlerjei, és más, handler tipusu rutinok, amelyek a felhasználói program futása, vagy annak egy része alatt rezidenssé tehetők.

Az előzőhöz hasonló probléma a táblázatok, buffer területek, munka területek kérdése. Vannak táblázatok, amelyek állandóan szükségesek, mások csak egy-egy perifériális átvitel ideje alatt. Átviteli buffer területre is csak meghatározott ideig van szükség, de a vezérlő terminálnak állandó buffer kell.

Az előzőekben az látható, hogy a program pillanatnyi igényeivel szoros kapcsolatban van a Supervisor számára egy adott időben szükséges terület mérete. Például minél több perifériálás átvitel van folyamatban egyidejüleg, annál több munkatábla lefoglalása szükséges abban az időintervallumban.

Ezen okok miatt célszerü a memória egy részét dinamikusan kezelni, amely területből szükség szerint ki lehet hasitani egy darabot, illetve fel lehet szabaditani egy lefoglalt darabját,

A TPA 70 operációs rendszerénél a következő memória terület kiosztást találtuk célszerünek:



1/a. ábra

1/b. ábra

Az 1/b ábra a dinamikus terület egy lehetséges állapotát mutatjuk meg, amely az egymás után beérkező igények hatására alakulhat ki.

A Supervisor rezidens területe nem kiván bővebb magyarázatot.

Az 1. overlay területen olyan rutinok váltogatják egymást, amelyekre a program igényeitől függetlenül is szüksége lehet a Supervisornak (konverziós rutinok, rendszer paramétereit szolgáltató rutinok, stb.).

A dinamikus területből hasitódik ki a 2. overlay terület ha a program olyan Supervisor szolgáltatásokat igényel, amelyek legtöbb programnál csak ritkán fordulnak elő (pl. file létrehozása, törlése, stb.), de viszonylag nagy méretü rutinok szükségesek kielégitésükhöz.

A felhasználói program területét és a dinamikus terület elválasztó szaggatott vonalról ejtsünk néhány szót.
A felhasználói program és a dinamikus terület határát a felhasználói program határozza meg egyrészt saját méretével, másrészt igényeivel. A program igényeitől függően kerülnek a memóriába, illetve törlődnek perifériák handlerjei, foglalódnak le buffer területek, munkaterületek. Tul sok ilyen igény esetén a dinamikus területből kihasitható terület elfogy, és csak operátori

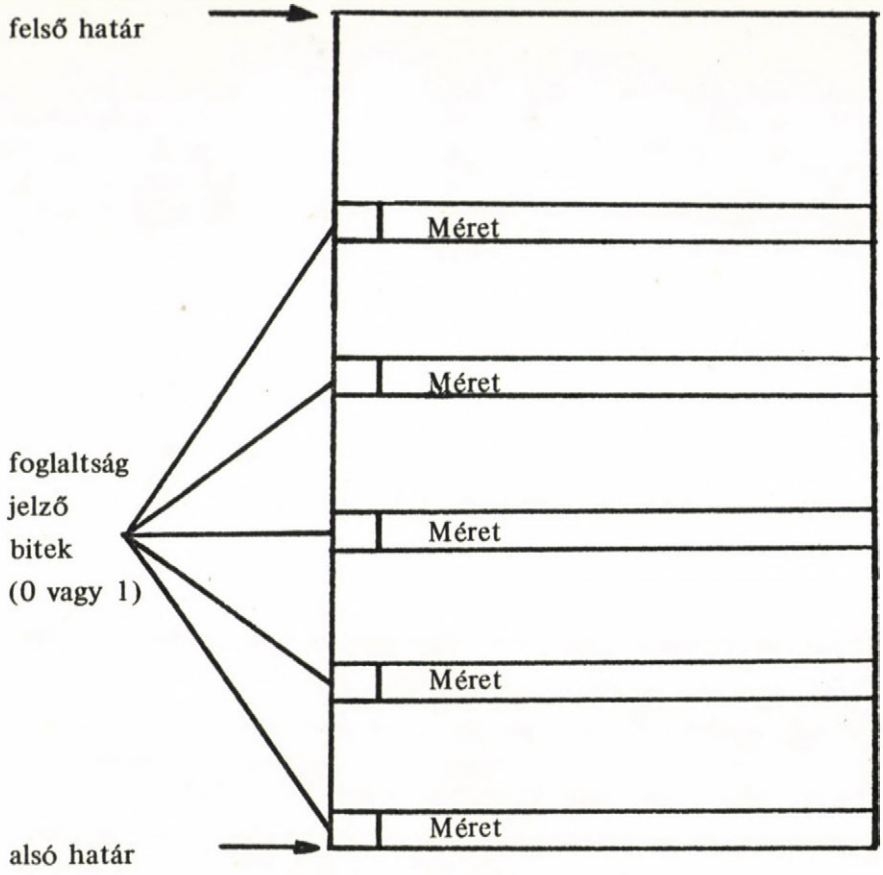
beavatkozással (pl. handler egy törlése) lehet a programot tovább futtatni. Ha ez nem megoldható akkor a program átalakitására van szükség (pl. overlay szervezés beépitése). A felhasználói program betöltésekor a dinamikus terület felső határa és a program alsó határa megegyezik. A programnak módja van ezen érték lekérdezésére, sőt szükség esetén igényelheti annak megváltoztatását is.
Ha a programozó nagy programot akar irni, akkor vigyáznia kell arra, hogy egyidejüleg csak a ténylegesen szükséges handlerek legyenek a memóriában, csak a szükséges file-ok legyenek megnyitva, és csak a szükséges bufferek legyenek lefoglalva. Ily módon növelheti saját programjának rendelkezésre álló területet a határ mozgatásával.

Fontos megjegyezni, hogy a tranziens rutinoknak és a dinamikus területre kerülő programoknak helytől független programoknak (position independent code) kell lenniük, hogy a memóriába bárhová betölthetők legyenek.

Szándékunkban van a Supervisor használata során statisztikákat vezetni arról, hogy egyes rutinokat milyen gyakran használnak. A statisztikák kiértékelése alapján módosithatjuk a rutinok csoportositását és esetleg több overlay területet vezethetünk be.
A dinamikus tároló terület kezelése:

A dinamikus tároló területet kezelő rutin memória rezidens. Mint minden ilyen rutinnál, ennél is nagyon fontos, hogy minél kisebb méretü legyen. Ez a követelmény azonban maga után vonja azt is, hogy egyszerü algoritmust kell alkalmazni.

A dinamikus területből tetszésszerinti méretü cella igényelhető, az előzetesen nincs felosztva semmiféle szempont szerint. A dinamikus tárolót kezelő rutin az alsó határtól kiindulva keres olyan összefüggő szabad területet, amely pontosan megegyezik az igényelttel. Ha éppen ekkorát nem talált, akkor a legnagyobb szabad területből hasitja ki a cellát, ilymódon elkerülhető, hogy a terület tulságosan felaprózodjon. A szükséges adminisztrációját a cellának a következő szóban végzi el. A dinamikus terület képe a következő lesz:



2. ábra

Terület felszabaditásakor csak a cella kezdőcimét kell megadni, és az szabaddá válik.

Ha már nincs elég nagy összefüggő szabad terület, akkor először az egymás mellett lévő szabad cellák összevonása történik meg, ha ez sem hozza meg a kivánt eredményt, akkor kerül csak sor a cellák elcsusztatására.
A fent leirt módon egy egyszerü, gyors rutin el tudja végezni a dinamikus tároló terület kezelését.

Az ismertetett módszernek külön előnye az, hogy akkor, ha a felhasználói programról tudjuk, hogy egyidejüleg a rendszer mely erőforrásait köti le, akkor becslés adható a feladathoz szükséges memória méretére, illetve adott memóriánál megadható a felhasználói program lehetséges maximális mérete.

S u m m a r y

Memory utilization of minicomputers

Marton Salamon

At the operating systems of minicomputers to utilize the core memory is particularly important. It is a general solutions, that beside a small resident part the transient moduls get a definite part of the memory. To decrease the number of disk transfers we must to fix in the memory certain routines (e.g. handlers) for a time period. The size of the buffers depends on the program too.

This paper describes a solution, wich takes the variable memory requirement of the user program in to account.

Р Е З Ю М Е

ИСПОЛЬЗОВАНИЕ ПАМЯТИ В МАЛЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИНАХ

М. Шаламон

В малых вычислительных машинах особенно важно экономичное использование операционной системой основной памяти. Общим решением этой проблемы является то, что рядом с маленькой резидентной частью транзиентные подпрограммы получают определенную часть памяти. С целью уменьшения вводов с диска, эти подпрограммы (например "handler") на некоторое время нужно считать резидентными. От программ пользователя зависит еще и размер буфферов.

В настоящей статье рассматривается решение, когда максимально применяется во внимание динамическая потребность в памяти программы использователя, работающей под операционной системой.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

SZÁMITÓGÉP RENDSZEREK TERVEZÉSE ÉS SZIMULÁCIÓS MODELLEZÉSE

Stauder Ernő
KSH Számitástechnikai Igazgatóság

1. A feladat megfogalmazása

A számitástechniaki szolgáltatásokat felhasználók gyakran teszik fel a kérdést, hogy miért tart olyan sokáig egy uj számitóközpont létrehozása, vagy egy már müködő központ jelentős bővitése. Kérdésük jogos, legalább is a felhasználó természetes kiváncsisága oldaláról. Hosszú ideje foglalkozik ezzel a kérdéssel a számitástechnikai szakemberek egy köre, akik részben vagy teljesen felelősek a számitóközpontok felállitásáért, üzemeltetéséért, vagy éppen annak vezetéséért. A hatvanas évek elején egy közepes számitóközpont létszáma 100-150 fő körül mozgott. A különböző szakosodások és a gépek teljesitményének növekedése eredményezte, hogy a hetvenes évek elejére ez a létszám közel kétszeresére nőtt mindenütt. A szakemberek egy része ezt természetesnek veszi, mert a korábbi évek "programozó centrikus" szervezeti felépitését – és amiből ez adódott, a feladatok ilyen értelmü tervezését – felváltotta egy igen sokszintü munkamegosztáson alapuló heterogén rendszer. A számitóközpontok, mint müködő rendszerek, a funkciójukat tekintve különböző feladatokat látnak el.

a.) **Szolgáltató központ**

A felhasználó a kivánt formában, az általa összegyüjtött és valamilyen módon rögzitett adatokból *feldolgozást* kér (és remélhetőleg kap). A felhasználók köre lehet véges, meghatározott – ekkor adott feladatok végrehajtására létrehozott számitóközpontról beszélünk. (tipikusak a termelő vállalatok saját számitóközpontjai). Ha a felhasználók körét nem korlátozzuk, ugy *bérmunka iroda* jellegü számitóközpontról beszélünk. Mindkét esetben jellemző, hogy a külső felhasználók szemszögéből az SzK egy *szolgáltató*, speciális feladatok elvégzését vállaló szervezet.

b.) **Termelő üzem**

A számitóközpontokban dolgozók ma már egyre jobban felismerik és tapasztalják, hogy a munkájuk milyen mértékben specializált és mennyire illeszkednie kell *időben és térben* a többi kapcsoló munkafolyamathoz. Bátran kimodhatjuk, hogy egy számitóközpont, saját müködését tekintve, egy üzemhez vagy gyárhoz hasonlitható. Speciális "terméket" állit elő, információt, azokból az adatokból, amelyet a megrendelő előirt számára. Még jobban közelithetjük a fizikai valóságot, ha azt mondjuk, hogy a megrendelő által rendelkezésre bocsátott vagy korábbi feldolgozásból származó és megőrzött adatok (megfelelő formában rögzitett) felhasználásával uj adatokat, illetve a korábbi adatok ujabb rendezett formáját állitja elő a számitóközpont. A "termelő folyamat" során a technológiát a feladat *szervezése* irja elő, az alkalmazott technológia pedig a rendelkezésre álló *software* függvénye. A felhasznált munkaerő tipusát a részfela-

dat jellege szabja meg: szervező, programozó, operátor, adatrögzitő, stb. A felhasznált eszköz pedig legtöbbször az SzK valamelyik *hardware* berendezése, vagy egyidejüleg több berendezése. Természetesen az ilyen "üzem" számos segédfolyamatot is igényel a "termelés" zavartalanságának biztositására, és ezeket ténylegesen (vagy ujtipusu fedő nevek alatt) meg is lehet találni minden számitóközpontban.

c.) Információs és adatraktár

A termelő üzem jellegnek a tárgyalásánál utaltunk a korábbi feldolgozásokból nyert adatok ismételt feldolgozására. Erre lehetőség nyilik azáltal, hogy az adatokat megfelelő tároló közeg – papir vagy mágneses alapanyagu – felhasználásával hosszú időre *tárolhatjuk*, megőrizhetjük. Az igy tárolt adatok gépi uton ismét beolvashatók. Ezek az adatok meghatározott felhasználóknak információkat jelentenek. Éppen ezért az adatok hosszú időre való tárolása egy speciális *raktározási* feladatot jelent a SzK számára. Ennek fontosságát már egyre jobban hangsulyozzák, különösen az adatok biztonságának kérdését. Ez elsősorban a meghatározott célra létrehozott számitóközpontok esetében igaz, de esetenként a bérmunkát végző SzK is elláthat ilyen feladatot, az ügyfelek egy körének.

Az előzőekben tulajdonképpen mindig ugyanarról a számitóközpontról beszéltünk, de különböző nézőpontról. Vizsgálatainakt a *müködő számitóközpontnak*, mint termelő egységnek az elemzésére koncentráljuk. Ezen belül is feltételezzük a *software* – technológia – adottságának és olyan modellezési módszereket keresünk, amelyek a számitóközpont terhelésének növekedésével összhangban biztositani képesek az erőforrás szükségletek megbizható becslését.

Különösen fontos a számitógép konfigurációjának megfelelő méretezése, mivel ez a legdrágább berendezés az összes között. Ezt egyfelől a várható munkák terhelése alapján méretezhetjük, másfelől az egyes konfiguráció elemek paraméterei alapján becsülhetjük az átbocsátóképességet. Érdekes lehet a SzK szellemi kapacitásának a becslése, teljesitmény és átbocsátóképesség alapján. A következőkben ezekre a kérdésekre igyekszünk választ adni.

2. A modellezés, mint tervezési módszer

Akár egy már üzemelő számitóközpont kapacitás bővitéséről, akár egy uj számitóközpontban üzemeltetendő uj berendezés méretezéséről beszélünk, minden esetben szükséges az elvégzendő feladatok lehető legpontosabb becslése. Ez a gyakorlatban legtöbbször a feldolgozások funkcionális egységeire (munkafolyamatok, munkalépések – azaz programok és programrendszerek) specifikusan értelmezett erőforrás szükségletek (CPU és) vagy start-stop idő, periféria igény) megadását jelenti. Természetesen ez a megadás már magában hordozza annak a konfigurációnak *specifikus* paramétereit, amelyet elképzelünk, vagy amelyen mértünk. A tervezési munka pontos végzésénél ezt mérlegelni kell) amit bizony sokszor "elfelejtünk"), és ennek egyik biztositéka lehet az adaptiv tervezési módszer. Ez a konfiguráció

azt jelenti, hogy a rendszer-független paramétereket igyekszünk kihangsulyozni és alapul venni, és az egyes méretezési részeredményeket az ujabb rendszerek (variánsok) kimunkálásánál már felhasználjuk. Természetesen továbbra is számottevő bizonytalansági tényező lehet a tervezés időpontjában a jövő – azaz esetleg a néhány év mulva értelmezett – számitógép rendszerének munkaterhelésének becslése. Ennek értelmezéséről, a méretezési feladat technikai kivitelezésében való befolyásáról itt nem kivánunk beszélni, mivel ez a módszertanról elterelné figyelmünket. A terhelést megadó feldolgozási paraméterek fentiekben megadott finomságu és léptékü ismerete azonban teszi a feladat tárgyát képező rendszer elég pontos lebontását. A számitógépet, mint rendszert olyan rendszerelemekből felépülő egésznek képezzük le, amelyhez illeszteni lehet a rendszer feladatát jelentő adatfeldolgozási munkákat, azok specifikus paramétereivel. Ez lehetővé teszi a teljes feldolgozási rendszer dinamikus modellezését, és a modell megfelelő müködtetése révén a méretezéshez szükséges rendszerparaméterek mérését. A becsült vagy kivánt paraméterek elérésénél rögzithetők a feldolgozási rendszerre jellemző alapértékek és ezeket tekinthetjük a számitógép konfigurációt leiró értékeknek.

A számitógép konfigurációnak az ismertetett módszertanon alapuló meghatározása elvezet a *makro-szintű rendszer modellezéséhez.* Ez várhatóan pontosabb tervezési horizontot tud majd biztositani, mint amit a hagyományos módszerek (általában elég sok és jelentősen torzitó becslést felhasználva) igérnek. A modellezés módszertanának bemutatására felhasználjuk a Gaver által kidolgozott makro-modellt [2], amelyt Hansmann és munkatársai módositott formában a gyakorlatban is felhasználtak rendszermérésre és méretezésre [3]. Hangsulyozni kell, hogy a modell alkalmazhatósága nem korlátozodik csak hosszutávu tervezésre, konfiguráció meghatározására, az alkalmazható rövid távu kapacitás tervezésre is. Nem látszik elég jónak speciális alkalmazási feltételek mellett, például az adatbázis orientált feldolgozások esetében. Erre más modellek állnak rendelkezésre. [1, 5] Természetesen ezek a modellek elméleti modellek, érvényességüket az adott lehetőségek mellett ellenőrizni kell, mielőtt elfogadnánk a tervezés módszertanaként.

A számitógép müködését tekintve megengedhető olyan egyszerűsités, amely az 1. ábrán látható strukturális elemekre bontja fel a rendszert. Ezután a feldolgozásra kerülő munkák, programok ilyen részekhez való hozzárendelése, illetve ezekből az erőforrásokból való igénylése szempontjából vizsgáljuk most ezt a modellt. A számitógép terhelését tekintve alkalmazzuk azt a megközelitést, hogy a terhelést a mindenkor a memóriában (tárolóban) található "program szegmensek" halmaza adja. Ezek a szegmensek minden esetben három különböző müveleten mennek keresztül: beolvassa az adatokat (input) a tároló adott részébe (partició, szegmens, stb.), bizonyos CPU időt használt fel a feldolgozásra, majd az eredményeket közli (output).
A modellben nem kerülnek megkülönböztetésre az alkalmazások vagy programok különböző tipusai. A terhelést kizárólag az I/0 és a CPU időráforditásokkal mérjük, minden egyes szegmensre.

A modellben feltételezzük a következőket:

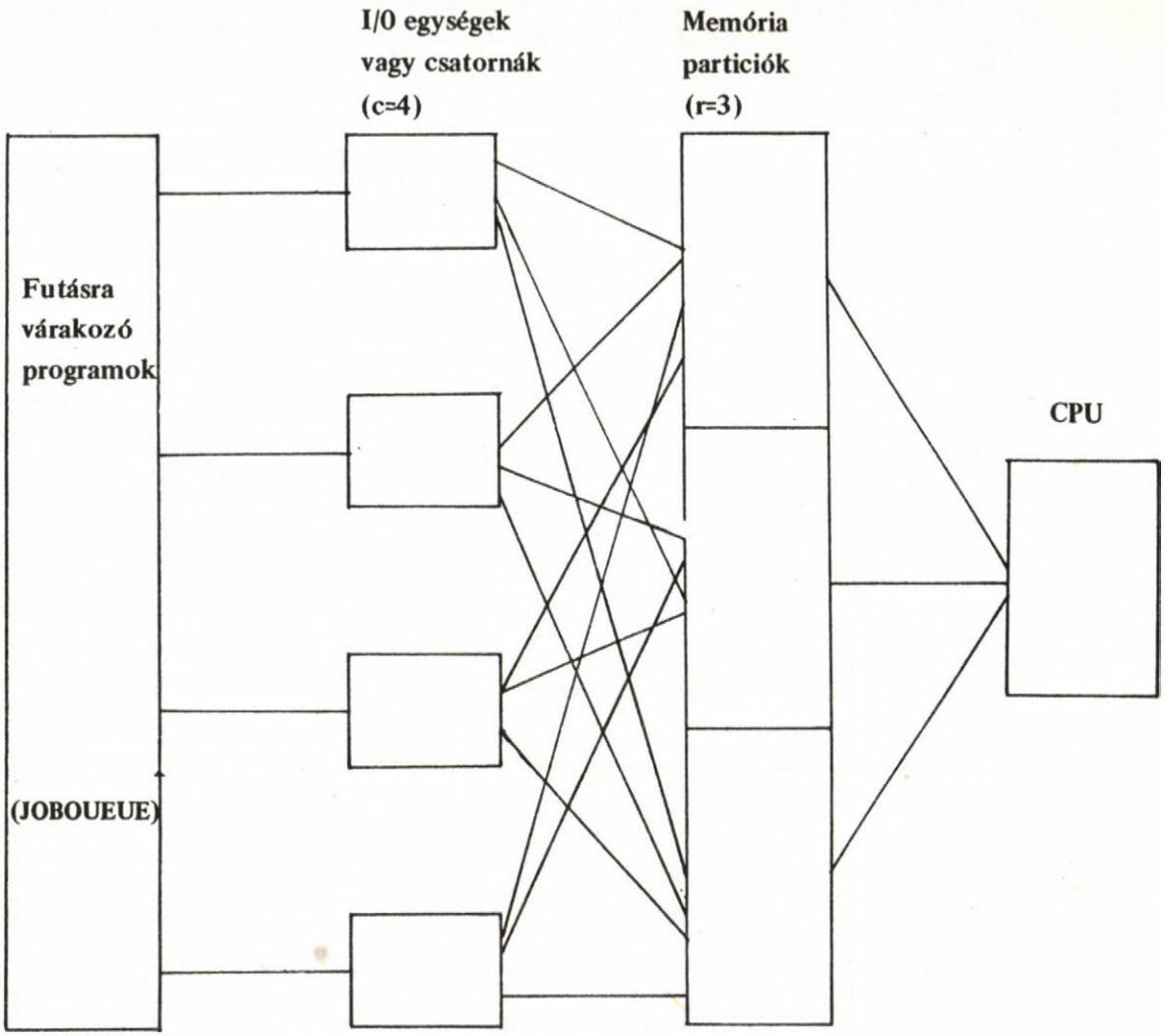
– A munkát, programokat a rendszer szegmensről szegmensre haladva dolgozza fel (hagyományos "lépésről-lépésre" módon);

- Minden szegmens egyetlen tároló terület (partició, stb.) és egyetlen I/0 egységet, illetve csatornát igényel;
- A felhasznált CPU idő szegmensenként változik, és az csak a véletlentől függ, az eloszlásról pedig feltételezzük, hogy exponenciális (a);
- A szegmensként felhasznált I/0 időről feltételezzük, hogy véletlen változó és eloszlásra jellemző, hogy exponenciális (b);
- A feldolgozás folyamatos, állandóan van feldolgozásra váró program;
- A központi tárolóban az egyes szegmensek által elfoglalt tárolóban az egyes szegmensek által elfoglalt tároló terület állandó nagyságu, és rögzitett. Igy a particiók számának növelése egyben a teljes memória kapacitás növelését is jelenti.

A fenti megszoritások eredménye az, hogy miután egy partícióhoz egyetlen csatorna (periféria) használatot feltételezünk, nincs várakozási idő addig, amig a csatornák (c) száma egyenlő vagy nagyobb a particiók (r) számánál. Továbbá az állandó munkaadagolás biztositja, hogy a csatornák felszabadulásával a partició ismét munkaképes lesz. Miután ezek a közelitések a valóságos feldolgozási rendszerkörnyezethez képest, korrekciós tényezők alkalmazásával pontosithatók a méretezési összefüggések.

A terhelés meghatározásához először vezessük be a a *foglaltság* fogalmát (f). Bár a szegmensek álatl felhasznált CPU és I/0 valószinüségi változó, mintavételek segeitségével meghatározható a CPU átlagos terhelése. A CPU foglaltsága befejeződik, ha a CPU nem talál ujabb olyan particiót, amelyre I/0 müvelet befejeződött. Eztuán a CPU *várakozik* (w). Ezeknek a felhasználásával kifejezhető a CPU átlagos terhelése.

(1) $$\overline{p} = \frac{\overline{f}}{\overline{f} + \overline{w}}$$



1. ábra: A Gaver modell szerinti struktura

A terhelésnek egy másik jellemzője lehet az átlagos számitási intenzitás. Ez a szegmenseknél CPU és I/0 idő arányát tükrözi. A [2] összefüggésben t_c és t_u valószinüségi változok, átlagértékeikkel származtatható az átlagos számitási intenzitás.

(2) $$\overline{\lambda} = \frac{\overline{t_c}}{\overline{t_u}}$$

Feltételezve a szegmensekben végzett munkák egymástól való függetlenséget, valamint exponenciális eloszlás szerinti viselkedést, igy a CPU átlagos terhelése csak a particiók és a csatornák számától függ, a következő összefüggés szerint:

(3) $$p = g(\overline{\lambda} \mid r, c)$$

Ebből tovább levezethetjük az egy particióra vetitett átlagos terhelés értékét:

(4) $$\overline{p}' = \frac{\overline{p}}{r}$$

Általánosságban értelmezhető az átlagos csatorna terhelés is, ha a CPU terheléssel közvetlen összefüggését elfogadjuk:

(5) $$q = \frac{\overline{p}}{\overline{\lambda} \cdot c}$$

Már itt szükséges és időszerü bizonyos korrekciós tényező bevezetése. Ugyanis a particiónak az állandó terhelést nehezen lehet biztositani, ezért állásidőre kerül sor a particiókon belül. (Vigyázzunk, ez nem azonos sem mérésben, sem jellegben azzal a várakozási idővel, amelyet a partícióban helyet foglaló program szegmens tölt el a következő I/0 müvelet befejezéséig!.) Ezért a modellben alkalmazott névleges partició számot (r), helyes egy effektiv partició számmal (r_e) figyelembe venni, ugyanakkor az egyes particiók is egynél több csatornát, illetve I/0 egységet igényelnek, igy erre is célszerü bevezetni egy effektiv értéket. Ezt egy korrekciós tényező bevezetésével érhetjük el az átlagos számitási intenzitás összefüggésében, és mondjuk azt, hogy az uj átlagérték legyen $\gamma\overline{\lambda}$. Ezzel a (3) egyenlet a következőképpen módosul:

(6) $$p = g(\gamma\overline{\lambda} \mid r_e, c)$$

Ez a modell most már alkalmasnak látszik a konfiguráció és a terhelés kölcsönös tervezésére, mind hosszutávu, mind rövidtávu becslések esetében.

3. A modell alkalmazása

Egy alkalmas kifejlesztett modell értékének, a gyakorlati alkalmazhatóságának ellenőrzésére Hansmann és munkatársai [4] számos mérést és elméleti számitást végeztek, különböző számitógép konfigurációk esetében. Gyakorlati mérések gerincét egy IBM 360/65 konfiguráción nyert adatok képezték, ahol,igen részletes adatgyüjtést végeztek a napi feldolgozásokról egy 16 órás termelési periodus során. A következő adatokat mérték:

- Partició fogalaltsági idő, a start-stop időkülönbség alapján;
- Program tipusok eloszlása;
- CPU produktiv idő az egyes szegmensek feldolgozásakor (az időegység 10^{-4} óra volt);
- CPU idő programonként;
- I/0 idő szegmensenként (ezt nem közvetlen méréssel nyerték);
- I/0 idő megoszlás a különböző periféria (csatorna) tipusokra (szintén közvetett méréssel nyerték).

Ezeknek az adatoknak a felhasználásával a számitási intenzitást, az effektiv partició számot és a CPU átlagos terhelését számitották, elsősorben a rövididejü szegmensek esetére. Ezeknek az adatoknak az alapján megkülönböztethető volt különböző partició vagy szegmens élettartamu feldolgozások, csoportja, különböző számu aktiv szegmens egy adott időben és ezek eloszlása, valamint az ezekhez tartozó I/0 idő felhasználás. Anélkül, hogy e részletes elemzésbe belemennénk, amely részben abból az alkalmazásból, részben az olyan tipusu adatgyüjtésből kinálkozik, néhány általános következtetést érdemes kihangsulyozni.

Adott számitóközpont jelenlegi terhelésének mérésére ma már igen sokféle részletes adatot tudunk gyüjteni, részben hardware, részben software mérési módszerekkel. A folyamatos mérés és adatgyüjtés lehetővé teszi a számitóközpont, illetve a konfiguráció terhelésének ellenőrzését, esetenként a terhelésének szabályozását. Az üzemeltetésről gyüjtött részletes adatok [8] esetenként közvetlenül felhasználahtók a várható terhelés becslésére, máskor pedig összetettebb statisztikai elemzések után alkalmazhatók, a korábban ismertetett modellnél input adatként. A rendszertervezési munkához a már müködő rendszerben mért adatok igen nagy segitségét jelentenek, mivel ezek támpontot szolgáltatnak egy uj konfiguráció különböző alternativáinak értékeléséhez.

Ezen az uton kiválaszthatók az elfogadható vagy elvetendő konfigurációs megoldások. A számitásokat a Gaver modell alapján végezhetjük, az ehhez szükséges adatokat pedig adott software monitor, például az IBM SMF/ (System Management Facility), alkalmazásával gyüjthetjük. Ezek után a módszer alkalmazható mind rövidtávu, mind hosszutávu tervezésre.

A rövidtávu terhelés tervezésénél könnyü dolgunk van, mert a várható struktura a módositás időpontjában valószinüleg hasonlitani fog a mérési időpontban tapasztalt program és feldolgozási strukturához. Igy számos konfiguráció lehetőségre alkalmazzuk a Gaver modellt és utána rangsoroljuk azokat, valamilyen kiválasztási politika szerint. Itt természetesen most már szállitási határidők, költségtényezők és egyéb, a számitóközpont szempontjából lényeges tényezőt is figyelembe kell venni.

Nem ilyen egyszerü az eset a *hosszutávu* tervezés esetében. Nagyon sokan kétségbe vonják egyáltalába hosszutávu tervezés létjogosultságát, különösen ilyen esetekben. A számitástechnika rohamos terjedése, szerepének vitathatatlan növekedése azonban nem engedi meg a polemizálást erről a témáról, hanem sürgeti az alkalmazható módszer bevezetését. Ezen a pon-

ton ismét hangsulyozni kell a modellezés fontosságát, mind módszertanilag, mint tervezéstechnikailag.

3.1. Az átbocsátóképesség vizsgálata

A SZK müködések egyik mércéje, hogy milyen volumenü feldolgozást tud valamilyen időegységre vetitve elvégezni. Mint *termelő üzem* a számitóközpont hasonlóan vizsgálható más termelő tevékenységet folytató vállalathoz. Azonban a termelés a folyamatában, az adatfeldolgozó teljes rendszerében igen különböző tipusu erőforrások kerülnek felhasználásra. Az egyensuly, vagy az optimum megtatálása nem minden esetben egyértelmü, különösen a költségek nem homogén jellege miatt. A tapasztalat azt mutatja, hogy a számitóközpontok nem alkalmaznak egységes elszámolási rendszert, ez pedig még mindig a különleges helyzetükből, a kinálatot meghaladó kereslet adta piaci helyzetből származik. Minden esetre a SZK szempontjából biztosan célszerü a minél több munka vállalása és remélhetőleg teljesitése. A terhelés maximális szinten tartása viszont megfelelő tartalék munkát, előre vállalásokat igényel. Ugyanakkor ezekre is reális átfutási időket kell meghatározni, hogy elfogadható teljesitési határidőt tudjunk adnia a megrendelőnek.

A teljes feldolgozási folyamatban az egyik kritikus pont maga a számitógép, amely a mai "harmadik generációs" világban a multiprogramozási környezetével igen sok lehetőséget kinál. Mint azonban már a 2. pontban utaltunk rá, igen nehéz bizonyos paraméterek becslése a feldolgozáshoz szükséges pontos átfutási idő meghatározásához, éppen az alkalmazott és software bonyolult müködésének részletes nyomonkövetése miatt. A matematikai és az operációkutatás területén kidolgozott uj módszerek, a valószinüségszámitás és a sorbanállási elmélet legujabb kutatási eredményei azonban biztató lehetőséget kinálnak a problémának a megoldására is. A számitógépek, aSZK átbocsátóképességének, a feldolgozások átfutási idejének becslésére már a 60-as évek közepén születtek modellek, szimulációra alapozott elemzések [6, 9, 17]. Ezek a batch-tipusu feldolgozási környezetet feltételezték alapvetően. Egy egészen speciális, de egyre jobban terjedő üzemeltetési környzet, a time-sharing (idő-osztásos) rendszerben történő feldolgozások paramétereinek becslésére megint más modelleket kellett kidolgozni [10, 11, 12]. Végül a 70-es évekre eljutottunk oda, hogy speciális programcsomagokat, illetve "nyelveket" fejlesztettek ki különböző számitógéprendszerek paraméteres elemzésére. [13, 15, 16]. Mivel a vizsgálat tárgyát képező rendszer önmagában igen bonyolult, következésképpen mindegyik rendszer valamilyen mértékü elhanyagolással él, makro szinten igyekszik lehetőséget adni a rendszer-elemzőnek paraméterei meghatározására.

Minden számitógéprendszer modellezése azonban önmagában hordja a modellező azon törekvését, hogy megszabja a szükséges információk körét és a vizsgálat szempontjából elégséges részletezettségét. Az "ideális" modellt tehát kettősség jellemzi:

a.) Képes a hardware berendezések széles skáláját és az operációs rendszerbelső logikáját együtt szimulálni, azok paramétereinek dinamikus változtatásával, a modell ujradefiniálása nélkül.

b.) Lehetővé teszi a számitógéprendszer különböző részeinek különböző részletességü definiálását, a feladat által megkivántak szerint.

A legtöbbször nehéz a modellt igy felépiteni, és esetenként az elsősorban az ésszerü egyszerüsitések miatt adódik. Kisebb szerepe van a modellezésben használt programnyelv, vagy a már meglévő rendszer adta korlátozó tényezőnek, de ezt is meg kell emliteni.

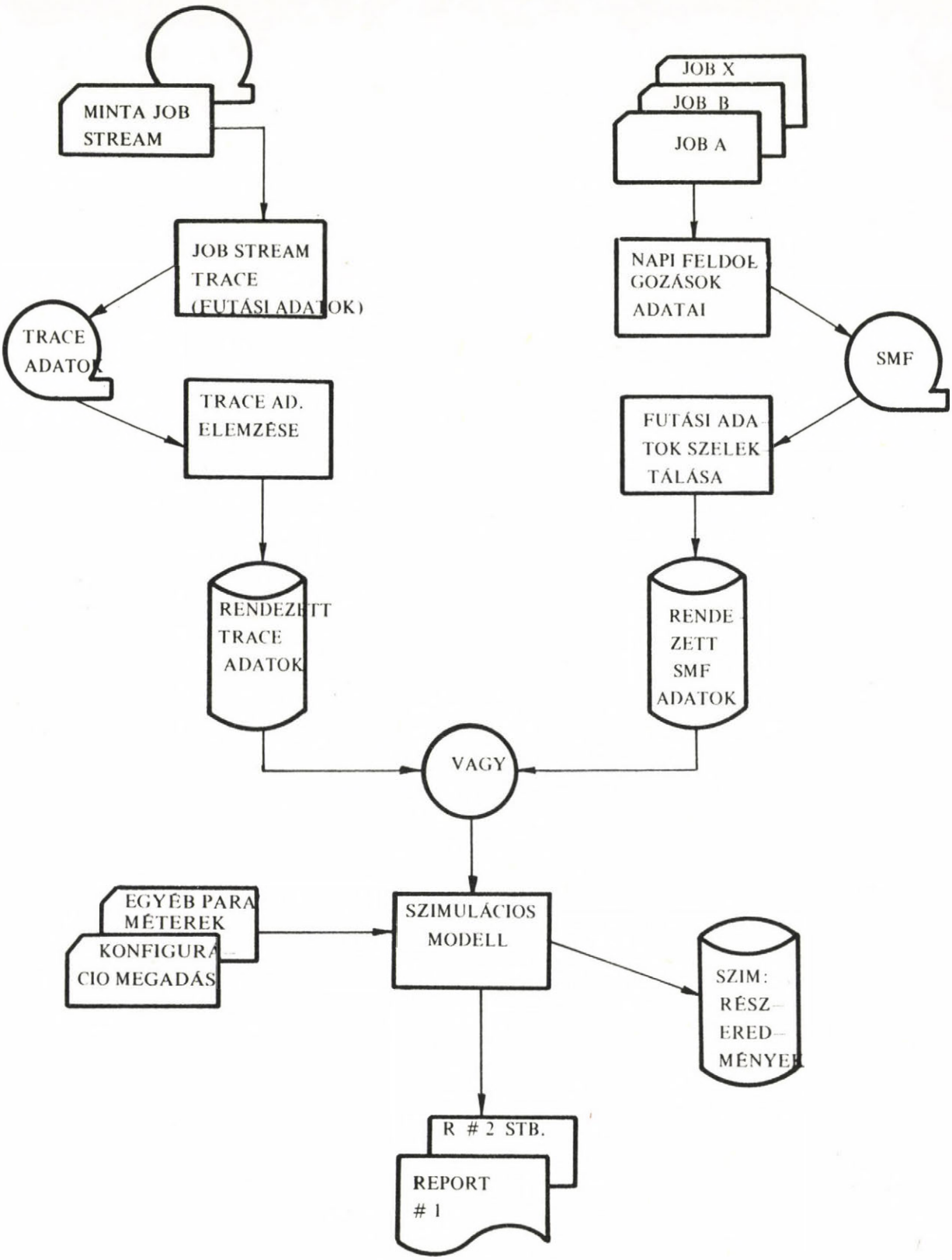
Abban az esetben, ha egy meglévő számitógép konfigurációja által átbocsátható programok számáról és tipusáról akarunk meggyőződni, megfelelő statisztikát kell készitenünk egy-egy periodus során az átbocsátott programokról és a terhelési struktura változtatásával elemezhetjük, hogy a különböző munka összeállitások esetében hogyan alakulnak a rendszer terhelési és az átbocsátott munkák időparaméterei. Ez a tipusu elemzés azonban nem tipikus.

Gyakoribb, talán a legtipikusabb vizsgálat az, amikor elég jól ismert a várható terhelés és kiváncsiak vagyunk, hogyan alakulnak a komplex rendszer különböző pontjain a mért értékek, ha a rendszer funkcionális összetevőit változtatjuk.

Miután a számitógép konfiguráció müködése jól meghatározott, annak belső változtatása hatással van egyéb rendszer pontokra is. Éppen ezek a hatások tisztázódnak, amelyre a rendszer bonyolultságából adódóan csak *teljeskörü* vizsgálat adhat választ. Ennek módszertana a modellezés, technikája a számitógépes szimuláció. Az ilyen rendszervizsgálatnál kétféle adatot kell inputként értelmezni.

Az első a vizsgált konfiguráció leiró adatok halmaza, külön-külön értelmezve az egyes modell vizsgálatokra. A másik a rendszer terhelését jelentő munkák, programok jellemző adatai. A konfigurációt leiró adatok általában statikusnak tekinthetők, néhány speciális esetben azonban szokás a tényleges paraméter értékét valószinüségi változónak tekinteni (pl. adott pillanatban a feldolgozási program által megcimzett sáv száma a mágneslemezen). A feldolgozási adatokat átlagértékként kezelhetjük, származtatásukra ismét a monitor által gyüjtött adatok, az abból képzett adatbázis szolgálhat. (Természetesen, ha még nincs számitógépünk, ugy ezeket az adatokat csak becsülni tudjuk. Lehetséges próba futtatások, próba üzemek végzése is hasonló konfiguráción, hogy legalább reális alapok legyenek a becslésre). A monitor által gyüjtött elemi adatok a következő csoportba sorolhatók:

- program betöltés (program inditás ideje;
- program befejezés ideje;
- állomány megnyitása (OPEN);
- állomány lezárása (CLOSE);
- csatorna (periféria kérés (I)0 tevékenység, tipus szerint);
- várakozás (WAIT);
- I/0 befejezésre várakozás /I/0 (INTERRUPT).



2. ábra A szimulációs modellezés folyamata

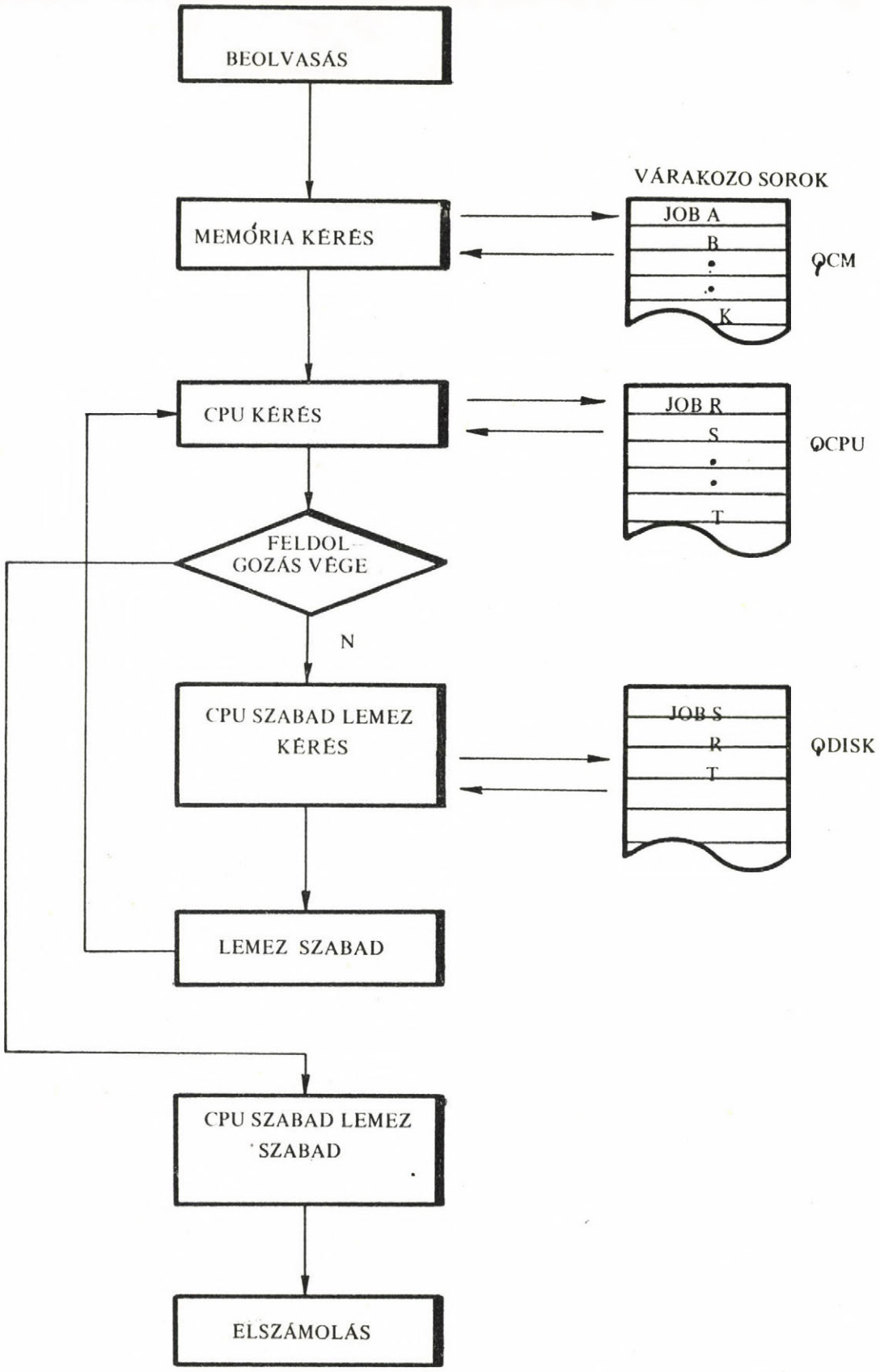
Ezeket az adatokat meghatározott időperiodus során mérjük, és rendezésükkel összeállítható egy tipikus terhelési struktura a vizsgált konfiguráció input job-folyamatként. A 2. ábra a modellezés általános folyamatát mutatja, az előbbi gondolatmenet alapján.

A szimulációs modellből egyrészt mérési eredményeket gyüjtünk a különböző konfigurációs pontokról, másrészt igyekszünk ujabb statisztikát felálltani az egyes munkák várható átfutási, várakozási és feldolgozási idejére. Ezután a modell változtatása nélkül, annak paramétereit vagy a terhelést változtatjuk – több munkát "pumpálva" a rendszerbe – és ismét gyüjtjük az eredményeket. Azok értékelése után, megfelelően rendszerezett formában már felhasználhatók közvetlenül a döntéshozatalhoz. Ez vagy a konfigurációra, vagy a vállat feladatokra összpontositva jelentkezik, de biztos, hogy a SZK optimális terhelésének irányába kell, hogy mutasson.

3.2. A számitógép konfiguráció komplex modellje: többcsatornás kiszolgálási rendszer

A 2. és 3.1 pontban egyaránt utalunk arra, hogy a számitógéprendszerek mai architektúrájának tekintve, sorbaállási rendszerként tekinthetők. Ezen az alapon kiindulva, a matematikusok először egyenletek és valószinüségi összefüggések sorozatával igyekeztek választ adni az ilyen rendszerek különböző viselkedésére. Bár a matematikai megfogalmazásoknál igen jelentős egyszerüsitéseket is alkalmaztak, illetve feltételeztek a rendszer viselkedéséről, számos összefüggés jól használható ezek közül, a különböző méretezéseknél [18]. A rendszerelemző azonban nem elégszik meg az átlagértékkel szereti tudni azt is, mi van az átlag mögött, melyik a szélső értékek. Erre a választ csak olyan dinamikus modellek felépitésével kaphatjuk meg, amelyek önmagukban hordozzák a sorbanállási rendszerek minden tulajdonságát. Ezt az elvet követjük akkor is, amikor a 3.1. pontban leirt modellezési technikát a gyakorlatban alkalmazzuk, ahogyan ezt az egyik kisérleti tervezésnél követték is [19]. A modell elméleti gyökere korábban vezethető vissza, [1], és a feldolgozást a következő elemi lépésekre bontja fel;

1. Program beolvasása a rendszerbe (várakozik szabad memóriára).
2. Amennyibe van szabad memória, betöltik a programot (t.i. az operációs rendszer).
3. A program feldolgozást (CPU) kér, ha nincs más program feldolgozás alatt. A CPU foglaltság a következő I/0 igénylésig tart.
4. Az I/0 kérésnél felszabaditja a CPU-t más munka számára. Ugyanakkor sorbaáll az I/0 teljesitéséért. Ha a sor üres – más program nem kérte ugyanazt a tipusu I/0-t – azonnal kezdetét veszi a kérés teljesitése.
5. Amint az I/0 befejeződik, a periféria (csatorna) szabaddá válik más program (I/0 kérés) számára. Ezután ismét a CPU kérésre kerül sor.
6. Amint a program befejezést nyer, felszabaditja a tároló területet egy következő munka számára.
7. A program elhagyja a rendszert.



3. ábra Egy program futásának belső feldolgozási lépései

A fenti lebontás természetesn sok egyszerüsitést is tartalmaz, de elég részletes ahhoz, hogy funkciónálisan a teljes számitógéprendszert vizsgálni tudjuk a feldolgozás oldaláról. A 3-4-5 folyamat annyiszor ismétlődik, ahány I/0 feldolgozási ciklus tartozik egy programhoz. A program feldolgozási lépései a 3. ábrán jól láthatók.

A feldolgozás legkisebb mért egysége a program (job), igy további részekre bontható, és ezek önmagukban "önálló életet" kezdenek élni, természetesen továbbra sem feledkezve meg "hovatartozásukról". Ugyanakkor uj kapcsolataik alakulnak azáltal, hogy más programok hasonló elemi lépései ugyanazt az erőforrást igénylik, azonos időben. Ekkor döntést kell hozni, "magasabb szinten", hogy melyik igénylő nyer kielégitést az erőforrással való kiszolgáláskor. Ez is jól látható a 3. ábrából, egyszerüsitett formában. Itt csak jelezhetjük a lehetséges várakozási sorokat: a memória, a CPU, a lemez, mint a leggyakrabban használt erőforrások, a kiszolgálásra várók sorából "táplálkoznak". Ahány tipusu erőforrást tudunk megkülönböztetni a rendszerben – és természetesen ezekre kiszolgálást kérni a programokból – annyi lesz legalább a rendszerben kezelt sorok száma. Tovább finomitható a rendszer általaz, hogy a perifériákat tipus szerint csoportositjuk, majd ezeket csatornákhoz rendeljük – ahogyan az a tényleges kofigurációkban is szerepel. Mivel egy csatornára többféle periféria (vagy csoport) köthető, esetenként ugyanaz a periféria (vagy csoport) több hardware csatornához is csatlakozhat, máris eljutottunk egy olyan bonyolult sorbaállási rendszerhez, amelynek vizsgálatára már semmiféle analitikus, matematikai formulákkal kezelhető módszer nem járható. Ehhez járul még a multiprogramozási környezetből adódó rendezett, dinamikus strukturált erőforrás kérések halmaza, amelyek együttes kezelése és értelmezése jelenti az *igazi* feladatot a modellezőnek, a szimulációit végzőnek. Látni kell azonban azt is, hogy egy ilyen modellezés igen költséges, még akkor is, ha már igen gyors gépek állnak rendelkezésre. (Érdekesség kedvéért megemlitendő, hogy az IBM erre a célra kifejlesztett programrendszere, s CSS [13] az első változatában a 360/40 gépen húszszor annyi időt használt fel, mint a szimulált periódus és 15 microsekundum periódus szimulálására a nagyon gyors 360/65 gépen is 0.24 millisekundumra volt szükség!). Éppen ezért nagyon oda kell figyelni, hogy a tervezés szempontjából milyen mértékig érdemes finomitani a rendszert – a tervezési költségek ésszerü korlátozása érdekében. Nem kétséges azonban, hogy már a számitógéprendszerek konfigurációs tervezését, a megfelelő feldolgozási követelmények pontos kielégitése érdekében, nem lehet csak egyszerü másolással, tipikus konfiguráció összeállitással elintézni. A SZK igen költséges üzem – mind önmagában, mind a felhasználóknak. Éppen ezért a tervezésre forditott idő és pénz megtérül, ha munkaigényesebb módszereket is kell használni.

I r o d a l o m

[1] P. H. Seaman, R. C. Soucy, Simulating operating systems. IBM Syst. J. 8, 4 (1969). 264-279.

[2] M.M. Mac Doughall, Computer system simulation: An introduction. Computer Survey 2, 3 (1970) 191-209.

[3] D. P. Gaver, Probability models for multiprogramming Computer Systems. Journal of ACM 14, 3 (1967) 423-438.

[4] F. Hanssmann, W. Kistler, H. Schulz, Modelling for computing center planning. IBM Syst. J. 10, 3 (1971) 305-524.

[5] B.R. Kirkerud, SIMBAS-A simple data base system. Norwegian Computing Center (1974) 19.

[6] M.I. Youchan, D.D. Rudie, E.J. Johson, The data processing system simulator (DPSS) Proc. AFIPS (1964) Fall Joint Computer Conf. 26, 251-276.

[7] G.S. Schedler, A queuing model of a multiprogrammed computer with a twolvele storage System. Comm. of ACM 16, 1 (1973) 3-10.

[8] W.I. Stanley, Measurement of system operational statistics. IBM Syst. J. 8,4 (1969) 299-308.

[9] L.R. Huesmann, R.P. Goldberg, Evaluating computer Systems throught simulation. Computer Journal 10, 2 (1967).

[10] N.R. Nielsen, The simulation of time-sharing systems. Comm. of ACM. 10, 7 (1967) 397-412.

[11] A. L. Scherr, An analysis of time- shared computer systems. MIT Press, Combridge, Mass. (1967).

[12] P.M. Seaman, On teleprocessing system desi design. Port VI. The role of the digital simulation. IBM Sys. J. 5,3 (1966) 175-189.

[13] Comutéer System Simualtion (CSS). H20-874-1 IBM DPD White Plarus, N.Y. (1972)

[14] E. Stauder, Program átfutási idők becslése szimulációs modellezéssel. Információ és Elektronika (1974) 9-14.

[15] ARTS-Analysis of Real–Time Systems. Cadis Project. University Stockholm (1972) 107.

[16] D.M. Braddock, C.B. Dowling, Simulation, Evaluation and Analysis Language (SEAL) IBM CPL.Prog. No. 360D 15.1.005

[17] G. K. Hutchinson, A computer center simulation project. Comm. ACM 8,9 (1975). 559-568.

[18] W. Chang, Single-server queueing processes in computing system. IBM. Syst. J. 9, 1 (1970) (1970) 36-71.

[19] Jo. Piene, Simulation of Computer Systems- Case Study. NCC pub. No. S 46. (1972) 113.

S u m m a r y

Planning and simulation modeling of computer system configurations

Ernő Stauder

The paper describes the simulation methods being able to evaluate and optimize the work of computer configurations.

Р Е З Ю М Е

ПЛАНИРОВАНИЕ И МОДЕЛИРОВАНИЕ
КОНФИГУРАЦИЙ СИСТЕМ ЭВМ

Е. Штаудер

В работе описываются методы моделирования, оценка и оптимизация работы конфигураций ЭВМ.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977.

ZUR BESTIMMUNG DER REAKTIONSZEIT VON PROZESSRECHENANLAGEN

Bergholz Gerhard

1. Die Prozessrechenanlage als Mittel zur Bedienung von Echtzeitauftragen

Eine wichtige Aufgabe von Prozessrechnern ist die zeitzyklische Überwachung von Ereignissen im technologischen Basissystem. Für diesen Fall soll hier die Reaktionszeit, die ein Mass für die Güte des Echtzeitbetriebs ist, bestimmt werden. Die Reaktionszeit für diese Überwachungsaufgabe ist die Zeit, die der Prozessrechner benötigt, um auf das Ereignis im technologischen Basissystem zu reagieren. Somit ist die Reaktionszeit T_{ri} eine Zeitdifferenz

(1) $$T_{ri} = t_{ri} - t_{ei}$$

wobei t_{ei} der Ereigniszeitpunkt im Basissystem und t_{ri} der Reaktionszeitpunkt des Prozessrechners sind (Bild 1).

Wir führen den Begriff Echtzeitauftrag ein. Dabei verstehen wir unter einem Echtzeitauftrag E_i die einmalige Bedienungsanforderung an die Prozessrechenanlage über das Unterbrechungssystem, wobei als Quelle des Echtzeitauftrags das technologische Basissystem oder die Echtzeituhr auftreten können. Wenn das technologische Basissystem Quelle des Echtzeitauftrags ist, sprechen wir von bedingungsabhangiger Auftragserteilung und wenn die Echtzeituhr zeitzyklisch Echtzeitaufträge erzeugt von zeitzyklischer Auftragserteilung.

Ein Echtzeitauftrag E_i besitzt eine Auftragsverweilzeit T_{vi}, die durch die Zeitdifferenz

(2) $$T_{vi} = t_{ri} - t_{ai}$$

bestimmt ist. Dabei ist t_{ai} der Ankunftszeitpunkt des Echtzeitauftrags E_i.

Die Überwachung von Ereignissen erfolgt gewöhnlich über eine zeitzyklische Messwerterfassung und – verarbeitung, wobei das entsprechende Programmsystem von der Echtzeituhr zeitzyklisch aktiviert wird.

In diesem Fall ist zwischen dem Ereignis im technologischen Basissysten t_{ei} und der Auftragserteillung t_{ai} keine Synchronität vorhanden und es entsteht eine Auftragsverschiebungszeit T_{si}, die durch die Zeitdifferenz

(3) $$T_{si} = t_{ai} - t_{ei}$$

ausgedrückt werden kann. Die Reaktionszeit besitzt somit bei zeitzyklischer Auftragserteilung zwei Summanden mit

(4) $$T_{ri} = T_{si} + T_{vi}$$

Bei bedingungsabhängiger Auftragserteilung gilt $T_{si} = 0$ und die Reaktionszeit ist gleich der Auftragsverweilzeit. Für einen Prozessrechnereinsatz werden in gewissen Zeitabständen ständing Echtzeitauftrage an den Prozessrechner erteilt. Bei zeitzyklisher Auftragserteilung erhalten wir dabei einen deterministischen und bei bedingungsabhangiger Auftragserteilung meist einen stochastischen Auftragsstrom. Da in den meisten Prozessrechnereinsätzen sowohl zeitzyklische als auch bedingungsabhängige Auftragserteilung vorhanden ist, müssen wir im allgemeinen einen stochastischen Auftragsstrom annehmen.

Für die Bestimmung der Verweilzeit eines Auftragsstromes ist die Ankunftsrate der Echtzeitaufträge von Bedeutung. Dabei verstehen wir unter Ankunftsrate eines Auftragstromes die Zahl der Echtzeitaufträge in der Zeiteinheite (hier s^{-1}).

In der Tafel 1 sind für ausgewählte Prozessrechnereinsatzfälle die Ankunftsraten λ, d.h. die Zahl der in der Zeiteinheit ankommenden Echtzeitaufträge in s^{-1} andegeben.
Bei der Analyse der Echtzeiteigenschaften einer Prozessrechenanlage interessieren die Reaktionszeiten der verschiedenen Echtzeitaufgaben [3,4,5] einer PRA. Somit ist es notwendig, die Echtzeitaufträge jeder Echtzeitaufgabe zu einem Auftragsstrom zusammenzufassen. Da wir die Reaktionszeiten als Zufallsgrössen auffassen können, ist es zweckmässig, für die Auftragsströme jeder Echtzeitaufgabe die Verteilungsfunktionen, bzw. die Erwartungswerte und Dispersionen der Reaktionszeiten zu ermitteln. Im Falle der zeitzyklischen Überwachung von Ereignissen werden somit alle Echtzeitaufträge, die zu einer Reaktion auf ein Ereignis im Basissystem führen, zu einem Auftragsstrom zusammengefasst.

2. Bedienungsmodell

Der Erwartungswert für die Reaktionszeit $E(T_r)$ ergibt sich aus den Erwartungswerten für die Auftragsverschiebungszeit $E(T_s)$ und für die Auftragsverweilzeit $E(T_v)$ mit

$$E(T_r) = E(T_s) + E(T_v) \tag{5}$$

Für die Auftragsverschiebungszeit gilt nach (8) für eine zeitzyklische Messwerterfassung der Grundzykluszeit Ta

$$E(T_s) = \frac{T_a}{2} \tag{6}$$

Zur Bestimmung der mittleren Auftragsverweilzeit kann als Modell der Prozessrechenanlage ein Bedienungsnetz [4, 5] betrachtet werden. Ein Bedienungsnetz kann mit Hilfe eines Bedienungsnetzplanes [5] und eines Forderungslaufplanes [4] und [5] graphisch dargestellt werden.
In Bild 2 ist der Bedienungsnetzplan für die Echtzeitaufgabe UNIMEP, das in [4] grob und in [7] ausführlich beschrieben ist, dargestellt. Der entsprechende Forderungslaupfplan wurde bereits in [4] angegeben.

Die Wechselwirkung zwischen den einzelnen Bedienungselementen erfolgt über geeignete Steueroperationen [3], die von Forderungsströmen [4, 5] aktiviert werden.

Die mittlere Auftragsverweilzeit kann aus den mittleren Verweilzeiten der Bedienungelemente $E(T_{vi})$ durch gewichtete Summenbildung

$$(7) \qquad E(T_v) = \sum_{i=1}^{n} q_i \alpha_{0i} E(T_{vi})$$

gewonnen werden.

Dabei sind i die laufenden Nummern, n die Zahl, α_{0i} die Intensitätsübertragungs-Faktoren und q_i die Parallelitäts-Koeffizienten der Bedienungselemente.

Die Intensitätsübertragungs-Faktoren geben an, wie sich die Forderungsstromintensitäten am Eingang des betreffenden Bedienungselementes gegenüber der Intensität des Ankunftstromes verändern [4] und [5].

Für die Echzeitaufgabe UNIMEP gilt

$$(8) \qquad \alpha_{0i} = \begin{cases} 1 & (i = 1,2,3,4) \\ M + 1 & (i = 5,6,7,8) \end{cases}$$

Bei der Ermittlung der Auftragsverweilzeit müssen wir berücksichtigen, dass bestimmte Bedienungselemente parallel bzw. quasiparallel einen Echtzeitauftrag bedienen. Das trift in unserem Fall für die Bedienungselemente AER, PUBE, ERFA einerseits und VERA andererseits zu, indem VERA parallel zur Messwerterfassung läuft. Bei parallelem bzw. quasiparallelem Betrieb von Bedienungselementen muss immer der Parallelzweig mit der grössten α_{0i} gewichteten Summenverweilzeit verwendet werden. Die Parallelitäts-Koeffizienten für die Echtzeitaufgabe UNIMEP ergeben sich zu

$$(9) \qquad q_i = 1 \qquad (i = 1,2,3,4)$$

$$(10) \qquad q_i = \begin{cases} 1 \text{ für } \sum_{i=6}^{8} E(T_{vi}), \geqslant E(T_{v5}) \\ \qquad\qquad\qquad\qquad (i = 6,7,8) \\ 0 \text{ für } \sum_{i=6}^{8} E(T_{vi}) < E(T_{v5}) \end{cases}$$

und

$$(11) \qquad q_5 = \begin{cases} \frac{1}{M+1} \text{ für } \sum_{i=6}^{8} E(T_{vi}) \geqslant E(T_{v5}) \\ 1 \text{ für } \sum_{i=6}^{8} E(T_{vi}) < E(T_{v5}) \end{cases}$$

Dabei wurde das Bedienungsnetz als probabilistischer Entscheidungsnetzplan [9] betrachtet und entsprechend umgeformt.

Die mittleren Verweilzeiten der einzelnen Bedienungselemente können aus den in [1] und [2] angegebenen Formeln für die mittleren Wartezeiten mit Hilfe von

$$E(T_{vi}) = E(T_{wi}) + E(T_{bi}) \tag{12}$$

berechnet werden. Dabei sind $E(T_{wi})$ die mittleren Wartezeiten und $E(T_{bi})$ die mittleren Bedienungsdauem der Bedienungselemente. Schliesslich erhalten wir für die mittlere Reaktionszeit genähert

$$E(T_r) = \begin{cases} \dfrac{T_a}{2} + (M+1)\left[E(T_{bb}) + \dfrac{E(T_{b8})}{1 - \dfrac{M+1}{T_a} E(T_{b8})}\right] & \text{für } \sum_{i=b}^{8} E(T_{vi}) \geqslant E(T_{v5}), \\ \dfrac{T_a}{2} + (M+1) - \dfrac{E(T_{bs})}{1 - \dfrac{M+1}{T_a} E(T_{bs})}, & \text{für } \sum_{i=6}^{8} E(T_{vi}) < E(T_{v5}). \end{cases} \tag{13}$$

3. Einige Untersuchungsergebnisse

Unter Nutzung der Formal (13) ergeben sich für die mittlere Zahl von Messwerten pro Grudzyklus die im Bild 3 dargestellten Abhängigkeiten der mittleren Reaktionszeit von der Grundzykluszeit der Messwerterfassung. Die Kurve setzt sich aus den beiden Zweigen $E(T_r)_1$ und $E(T_r)_2$ zusammen, wobei $E(T_r)_1$ der ersten Zeile und $E(T_r)_2$ der zweiten Zeile auf der rechten Seite von Formel (13) entspricht.

Wir erkennen deutlich, dass es in Abhängigkeit von der mittleren Grundzykluszeit eine minimale mittlere Reaktionszeit gibt.

Dieses Minimum folgt daraus, dass die mittlere Auftragsverweilzeit mit wachsender Grundzykluszeit fällt und die Auftragsverschiebungszeit mit wachsender Grundzykluszeit wächst. Diese Tendenz tritt auch auf, wenn in (13) kein Umschlag von der ersten zur zweiten Zeile eintritt. Der Anwender von Prozessrechnern berücksichtigt häufig intuitiv nur die Auftragsverschiebungszeit und fordert eine möglichst kleine Grundzykluszeit. Diese Forderung ist, wie das Bild 3 zeigt, nicht immer gerechtfertigt. Bei Verringerung der Grundzykluszeit T_a über das Minimum der mittleren Reaktionszeit hinaus, wird die mittlere Auftragsverweilzeit so gross, dass die mittlere Reaktionszeit sehr schnell gegen ∞ strebt.

Um ein umfassenderes Bild über die Reaktionszeit zu bekommen, ist es zweckmässig, die Verteilungsdichtefunktion der Reaktionszeit zu bestimmen. Diese wurde für die Echtzeitaufgabe UNIMEP durch stochastische Simulation mit Hilfe des verfahrensorientierten Programmiersystem SIMDIS für die Grundzykluszeit $T_a = 1$ un die mittlere Messstellenzahl pro Grundzyklus $M = 100$ ermittelt.

In Bild 4 ist das Ergebnis der Simulation als Balkendiagramm dargestellt. Ausserdem enthält das Bild die Gamma-Verteilungsfunktion, welche für den Mittelwert und die Standardab-

weichung berechnet wurde, die sich aus der Simulation ergab.

Es ist zu erkennen, dass die Verteilungsdichtefunktion einer Gamma-Verteilung ähnlich ist.

Der bei der Simulation ermittelte Wert der mittleren Reaktionszeit $E(T_r) = 1{,}135$ bestätigt, dass die theoretische Formel (13) für die betrachteten Werte von T_a und M mit $E(T_r) = 1{,}244$ durchaus befriedigende Ergebnisse liefert.

Literaturverzeichnis

[1] G. Bergholz, Modell für die Untersuchung des Echtzeitverhaltens eines Prozessrechners. mrs. 18 (1975) H. 4.

[2] G. Bergholz, Echtzeitmodell für den Informationsaustausch zwischen der Zentraleinheit und der Parpherie eines Prozessrechners. msr 18 (1975) H. 11

[3] G. Bergholz, Recnergestützte Gestaltung von Prozessrechner Anwendungssystemen. msr 18 (1975) H. 5, H. 9.

[4] G. Bergholz, Zur Ermittlung der Forderungsstromintensitäten in einem Echtzeitoperationssystem für Prozessrechenanlagen. Wiss. Z. Techn. Universität Dresden 24 (1975) H. 3/4

[5] G. Bergholz, Zur Analyse der Multiprogrammbearbeitung in einer Prozessrechenanlage. Rechentechnik/Datenverarbeitung, 11 (1975) Beiheft 3.

[6] G. Bergholz, Modell für die Untersuchung des Echtzeitverhaltens eines Prozessrechners. XVIII. Intern. Wiss. Koll. Th Ilmenau 1973. Band B 3

[7] S. Donat, Beschreibung des Standardprogramms UNIMEP Systemunterlagen-Dokumentation POS PRS 4000/KRS 4200 v. 30.4. 1973

[8] L. A. Owtscharow, Anwendungsaufgaben der Bedienungstheorie (russich) Verlag Maschinostrojenije Moskau (1969)

[9] K. Renger, Entscheidungsnetzpläne – Instrumentarium der modernen Wissenschaftsorganisation. Rechentechnik/Datenverarbeitung, 7 (1970) H. 10.

Összefoglaló

Processzorok reakcióidejének meghatározásához

Gerhard Bergholz

Ebben a cikkben egy programrendszer reakcióidejével foglalkozunk egy adott technológiai rendszer ciklikus felülvizsgálatával végzett mérési adatfelvétel és – feldolgozás esetén.

A közepes reakcióidő meghatározása analitikusan következik egy kiszolgálási modellből, ahol a reakcióidő, mint a rendszer egyes elemei várakozási idejeinek sulyozott közepeként adódik. A cikk kvalitativ állitást tartalmaz korlát megadásához a mérési adatok felvételének (alap -) ciklusideje korlátozásához.

Az analitikus vizsgálatokat kiegészitően a SIMIDIS programozási rendszer segitségével végzett sztochasztikus szimuláció eredményeit is bemutatja.

Р Е З Ю М Е

ОБ ОПРЕДЕЛЕНИИ ВРЕМЕНИ РЕАКЦИИ ПРОЦЕССОРОВ

Г. Бергхольц

В настоящей работе излагается метод определения системы программ для измерения и обработки данных некоторой технологической системы.

Среднее время реакции определяется аналитически на основе модели массового обслуживания, в котором время реакции является взвешенным средним времени ожидания отдельных элементов системы.

В работе дается качественная оценка для ограничения времени основного цикла измерения. Кроме аналитических исследований, вопрос изучался и методом статистических испытаний с помощью системы программ .

Lfd.Nr.	Einsstzfall	Ankunftstrate
1	PVC-S-Produktion	0,200
2	Werkzeugmaschinensteuerung	0,568
3	Kaltband-Tandemvalzstrasse	0,988
4	Kaliaufbereitungsbetrieb	1,014
5	Klinisches Labor	3,121
6	Patientenüberwachung	250,000

Tafel 1. Ankunftsraten verschiedener Prozessrechnereinsatzfalle

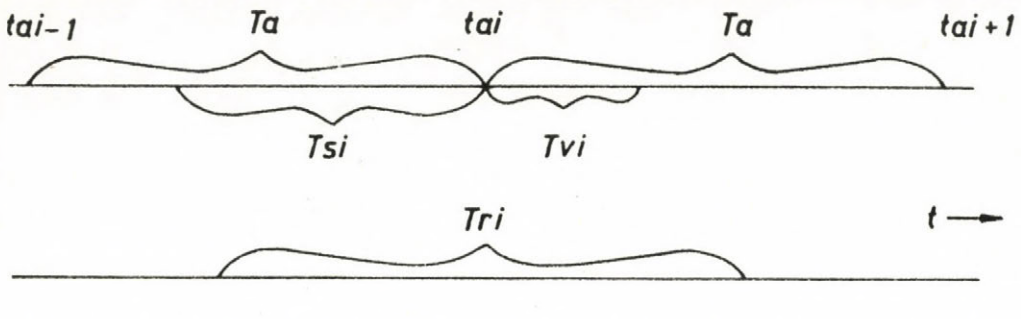


Bild 1 Zur Definition der Reaktionszeit bei zeitzyklischer Überwachung von Ereignissen

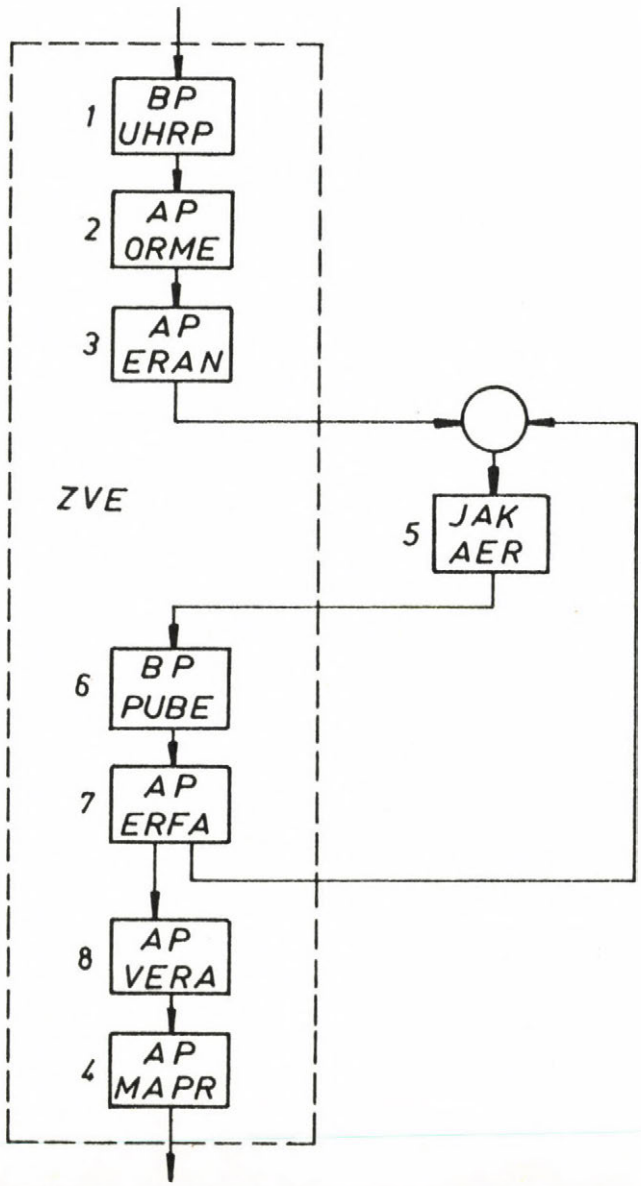


Bild 2: Bedienungsnetzplan für den Auftragsstrom der Echtzeitaufgabe UNIMEP

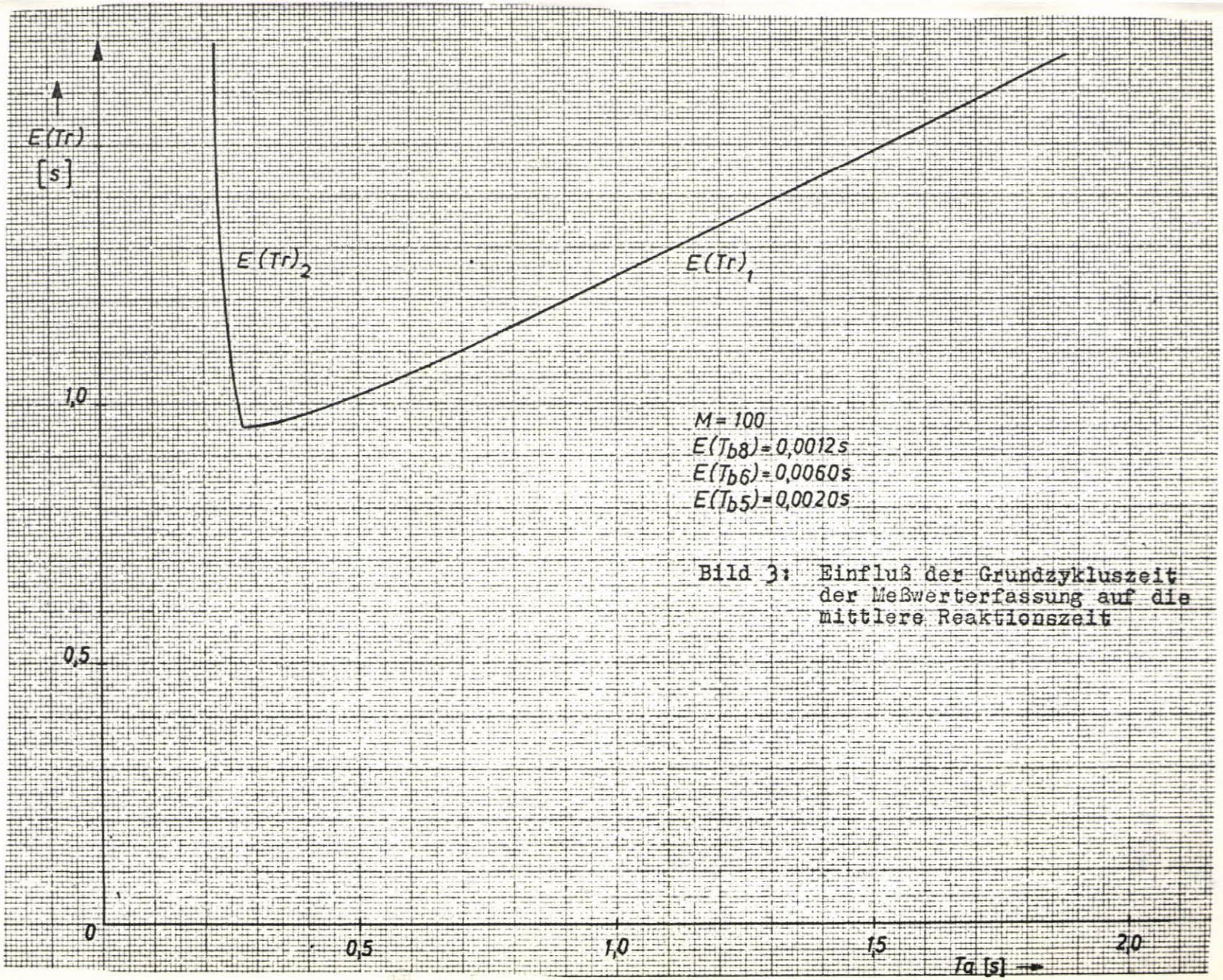


Bild 3: Einfluß der Grundzykluszeit der Meßwerterfassung auf die mittlere Reaktionszeit

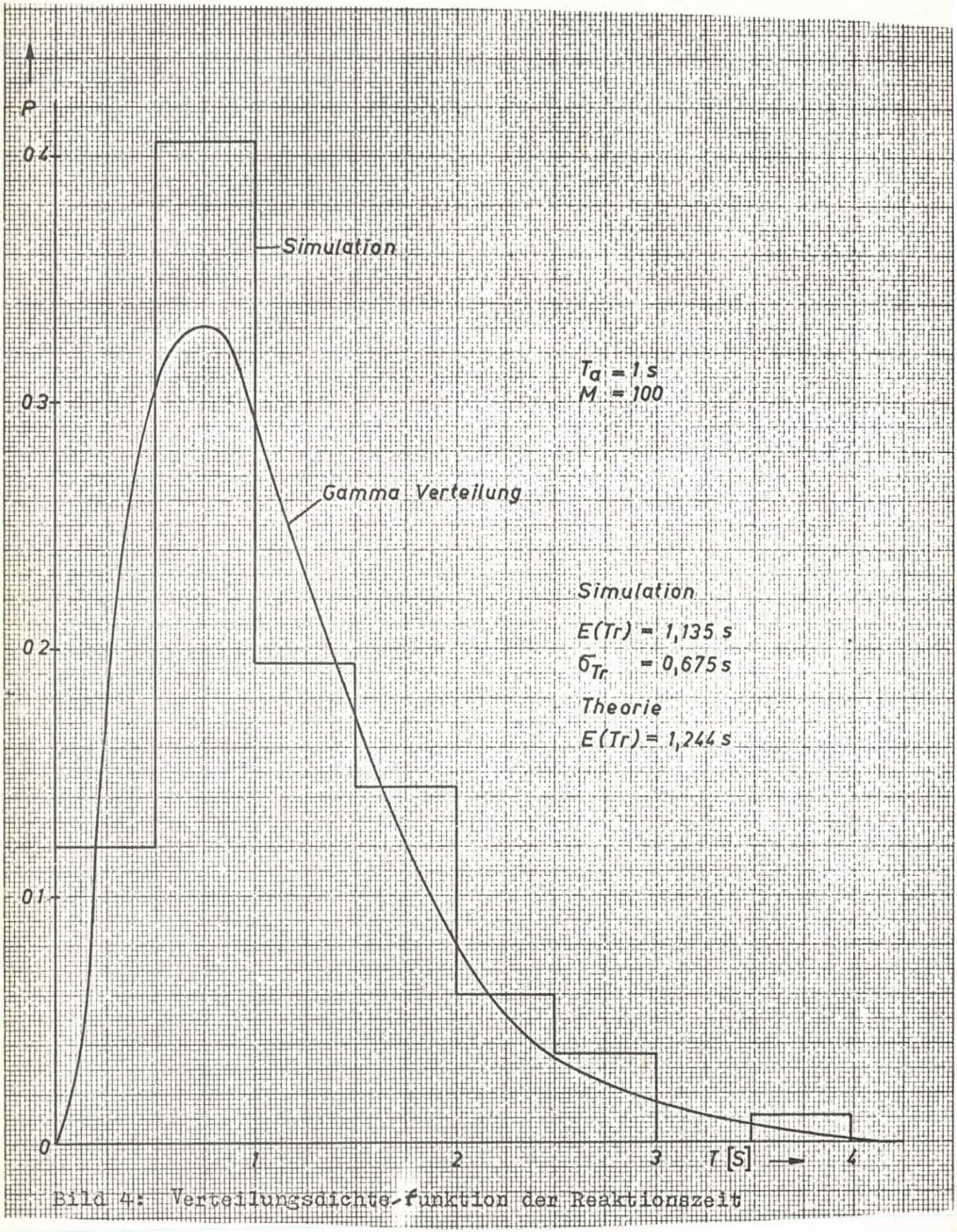


Bild 4: Verteilungsdichte-funktion der Reaktionszeit

MTA Számitástechnikai és Automatizálási Kutató Intézete, Közlemények 18/1977,

DINAMIKUS ERŐFORRÁS ELOSZTÁS VEGYES REAL–TIME ÉS BATCH–ÜZEM ESETÉN

Gyarmati Péter
KSH Számitástechnikai Igazgatóság

1. Bevezetés

A számitástechnika mai állása lehetőséget nyujt arra, hogy a különböző feladatokat a nekik megfelelő legalkalmasabb üzemmódban hajthassuk végre. A különböző üzemmódokat legcélszerübben külön-külön számitógépeken lehetne megvalósitani, azonban erre kereskedelmi, illetve gazdasági okok miatt általában nincs mód. Mivel korunk számitógépének kapacitása, erőforráskészlete, sebessége igen nagy, meg van a lehetőség arra, hogy a különböző üzemmódokat egyazon gépen valósitsuk meg.

A távfeldolgozási igény egyre jobban terjed, de még ma is jelentős mennyiségü a hagyományos módon végzett feldolgozás. Ennek a kettős igénynek a kombinációja adja a vegyes, real-time és batch üzem felállitásának szükségességét. Feltéve azt, hogy már rendelkezünk a vegyes üzemmódhoz szükséges, alapvető hardware és software eszközökkel, akkor még mindig biztositanunk kell a beérkező feladatok párhuzamos, egyidejü és ami a legfontosabb – igény szerinti futtatását.

Ennek a problémának a következő – egymásnak potenciálisan ellentmondó – teljesitmény tényezői vannak:

- a real–time üzemhez szükséges válaszadási idő (response time);
- a batch üzemü feladatok átfutási ideje (turnaround time);
- a rendelkezésre álló erőforrások minél jobb kihasználása a költségek csökkentése érdekében (throughput).

Az általában rendelkezésre álló ütemezési algoritmusok egy-egy tényezőt kielégithetnek, de többnyire ellentmondanak a többi tényezőnek.

Például egy maximális erőforrás kihasználásra törekvő algoritmus teljesen megváltoztathatja, tetszőlegesen hosszura növelheti a real-time üzem válaszadási idejét.

Mivel a rendszer tervezésekor általában nem ismerjük teljes pontossággal az igényeket, az üzemet csak valamilyen valószinüséggel tervezhetjük meg, amelyhez képest a ténylegesen állapotok elérhetnek.
Különösen vonatkozik ez a csúcs terhelésü időszakokra.

Célszerü tehát egy rugalmasan változtatható, változó, dinamikus erőforrás elosztási algoritmus kialakitása, amely a valószinüségi alapon tervezett rendszert igyekszik a pillanatnyi igények szerinti optimális módon szabályozni.

Az előadás célja egy ilyen dinamikus mérő és szabályozó algoritmus tapasztalatokon alapuló, logikus megközelitése.

2. A cél kitüzése

Először rögzitsük a batch, illetve a real-time feldolgozás rövid definicióját. Ez a definició nem lesz teljesen pontos, de elegendő az előadásban felmerülő kérdések megértéséhez, tárgyalásához. A definició készithető akár az idő, akár a feldolgozandó adat alapján.

Definició a feldolgozandó adat alapján

Ha a feldolgozási igény az adat keletkezésével egyszerre lép fel, akkor *real-time* feldolgozásról beszélünk.
Ha az adatokat elöbb adatállományba összegyüjtjük, majd egyszerre dolgozzuk fel, akkor *batch* feldolgozásról beszélünk.

Definició az időre alapozva

Ha a feldolgozás ugyanabban az időben és időléptékben megy végbe, mint a feldolgozással modellezett valóságé, akkor *real-time* feldolgozásról beszélünk.
Ha a feldolgozás ideje és időtartalma a modellezett valóságtól független, akkor *batch* feldolgozásról beszélünk.

A két feldolgozási mód alkalmazása egyazon gépen nem uj probléma, de a mai fejlettségi állapot egy ujabb megközelitést tesz lehetővé.

A vegyes üzemre történő átálláskor a legalapvetőbb alrendszereket át kell értékelni, esetleg teljesen ujra kell tervezni.
Ezek:

- az adatok tárolásának és hozzáférésének rendszere;
- az adatgyüjtés és adatbevitel;
- a feldolgozási igények kezelése és irányitás;
- a számitógép kezelése és üzemeltetése.

Témánk az utóbbi két alrendszerrel kapcsolatos, pontosabban fogalmazva egy olyan *algoritmus felállitása a számitógép üzemeltetésére, amely a feldolgozási igények kezeléséből fakadó szempontokat a lehető legjobban elégiti ki.*
A cél tehát, hogy elfogadható határok között tartsuk

- a beatch jobok átfutási idejét;
- a real-time utasitások válaszadási idejét;
- a számitógép legkedvezőbb kihasználását;

az erőforrások elosztásának *igény szerinti* irányitásával. A megvalósitáshoz abból az állitásból kell kiindulnunk, hogy a felhasználói igények és a számitógép üzeme egymástól független, annak

ellenére, hogy a gép konfigurációját és az üzemi paramétereit egy adott időszak felhasználói igénye alapján igyekszünk megvalósitani.

Különböző kompromisszumokat, az időt és a felhasználói igények változását figyelembevéve mindig kiderül, hogy a tervezett rendszer végül fázis késésben van. Ha a rendszerünket sikerül elegendően adaptivvá tennünk az emlitett változásokra, akkor a fázis késés – a kapacitás problémáján kivül – megszüntethető. Az adaptiv vezérlés egyik módja az erőforrások felhasználói igény szerinti minél dinamikusabb elosztása. Elvileg minden idő pillanatban az erőforrások ugy oszlanak meg, hogy mindig a legjobban "rászorult" igények jussanak hozzá.

Ehhez két fontos, alapvető tényező szükséges:

– az erőforrások megfelelő eloszthatósága;

– a felhasználói igények pilllanatnyi "rászorultságá-"nak ismerete.

A két tényező ismeretében az algoritmus már nagyon egyszerü:

1.) Megállapitandó a felhasználói igények pillanatnyi "rászorultsága".

2.) Az erőforrások elosztása a "rászorultság" rendjében.

3.) Ha elértük a kapacitás határát a "legkevésbé rászorultakat" késleltetjük és visszatérünk az 1. pontra.

3. Az erőforrások eloszthatósága

A dinamikus algoritmus megvalósitásához a gép erőforrásainak dinamikusan ujra eloszthatónak kell lenniök, tehát olyan számitógépre és operációs rendszerre van szükség, amely ezt lehetővé teszi.

Ennek vizsgálata a jelen előadásnak nem célja, tehát feltételezzük, hogy ez teljesül a legfontosabb erőforrások – CPU idő, input-output utasitások, központi tár – quantitativ mérési lehetőségével együtt.

4. A felhasználói igények pillanatnyi "rászorultsága"

A rendszer tervezési szakaszában a felhasználói igények ismeretében különböző *teljesitmény-csoportokat* állitunk fel, amelyeket erőforrás felhasználás szempontjából különböző küszöbértékekkel és paraméterekkel jellemezhetünk. Ha futtatás közben a felhasznált erőforrásokat hasonló értékekben mérjük, akkor ezek összevetésével megállapitott eltérés egy olyan mennyiséget ad, amely jellemző az igény pillanatnyi erőforrás "rászorultságára" és igy a teljesitmény-csoportjának megfelelő kiszolgálásban részesülhet.

A mérés és a vizsgálat elvégzéséhez az alábbi definiciókban leirtak szükségesek.

4.1. **Definició.** Kiszolgálási mérték (S)
Egy adott időintervallumban a felhasználói igény által használt erőforrások sulyozott mennyisége időegységre vonatkoztatva,

$$S = A(CPU) + B(MEM) + C(i/0)$$

– A, B, C rendszerparaméterek az erőforrások sulyozására.

– Az erőforrások mennyiségi mérése sokféleképpen oldható meg.

Egy példa:

$$(CPU) = \frac{1}{t} \cdot \frac{\sum_t t_{cpu}}{t_{const}}$$

ahol $\sum_t t_{CPU}$ a t időintervallumban felhasznált *CPU* idő és t_{const} az egységnyi *CPU* idő, pl: 1000 átlagos *CPU* utasitás végrehajtásához szükséges idő.

$$(MEM) = (CPU) \cdot n_{page}\Big/_t$$

ahol n_{page} a t időintervallum végén aktiv központi tároló lapok száma.

$$(i/0) = \frac{1}{t} \sum_t n_{command}$$

a t időintervallumban végrehajtott $i/0$ utasitások száma.

4.2. **Definició.** Mérési időintervallum (t)

A dinamikus algoritmus egyes végrehajtásai között eltelt idő. Az egyes időintervallumok különböző hosszuságuak lehetnek, kezdetét, vagyis az algoritmus végrehajtását az alábbi események határozzák meg:

– CPU várakozó állapotba kerül:

– page fault (központi memória lapváltás igénye):

– uj igény belépésekor.

4.3. **Definició.** Teljesitmény mérték (P)

Minden teljesitmény-csoporthoz tartozik egy teljesitmény mérték, amely szerint az ide belépő felhasználói igény futtatásakor erőforrást kér. A kiszolgálási mértékkel azonos léptékü és dimenzióju.

4.4. **Tétel.**

1. $P < S$ *a kiszolgálás magasabb szintü az igenyeltnél, KÉSLELTETHETŐ!*
2. $P = s$ *a kiszolgálás megfelel az igénynek*
3. $P > S$ *a kiszolgálás alacsonyabb szintü az igényeltnél, KÉSIK!*

5. **Algoritmus.**

Minden feldolgozási igény a három döntés alapján kerül besorolásra a következő időintervallum idejére az alábbi módon:

1. Az igény a *várakozó sorba kerül* (SWAP OUT) jelzést kapja az (S – P) érték nagyságának növekvő sorrendjében.
2. Az igény a *kiszolgálás folytatódik* jelzést kapja, ha a várakozó sorban volt, akkor az *aktiválásra kerül* jelzést kapja (SWAP IN) a várakozó sorban elfoglalt helyének sorrendjében.
3. Az igény a *kiszolgálás folytatódik a 2. szerinti igényeket megelőzve* jelzést kapja a (P – S) érték nagyságának csökkenő sorrendjében; ha a várakozó sorban volt, akkor az *aktiválásra* (SWAP IN) *kerül* jelzést kapja a (P – S) érték nagyságának megfelelő besorolással.

Az ezután következő *SWAP algoritmus* fogja az igényeket a központi tár, illetve a virtuális tár lehetőségeinek megfelelően elhelyezni.

A SWAP algoritmus a tárkapacitástól függően a kiszolgálás folytatódik jelzésüeket is a várakozó sorba helyezhet, illetve a várakozó sorba kerül jelzésüeket is aktivan hagyhat.

Az algoritmus végsősoron a felhasználói igényeket minden egyes időintervallum végén a kiszolgálástól függően – az (S – P) érték nagysága szerint – sorba rendezi, majd a SWAP algoritmus a tár kapacitásának megfelelő számu – a sorban elől álló – igényt aktivál.

Néhány megjegyzés az algoritmushoz

Nem feltétlenül jelent túlterhelést, ha vannak igények a várakozó sorban (SWAP OUT)

A pillanatnyi túlterhelés nem biztos, hogy késést okoz a külső ismérvek (response, turnaorund time) alapján.

A rövid idejü túlterhelések kiegyenlitődnek, mert mindig azok kerülnek késleltetésre, amelyek a legjobban elviselik.

A rendszer hatékonysága akkor érvényesül, ha a belépő összes igény azonnal aktiválásra kerül. Ennek csak a SWAP algoritmus gazdaságossága szab határt.

Az algoritmus lehetőséget ad az optimális gépkihasználásra, mivel csak az időintervallum határokon lép be és csak a feldolgozási igények végrehajtási sorrendjét változtatja meg.

A megoldás a SWAP algoritmus szempontjából nem optimális, azaz lehetséges olyan helyzet, amikor feldolgozási igények sokszor mozognak a központi tár és a várakozó sor között (SWAP–IN, SWAP–OUT).

A késleltetések, illetve a késések elosztása az egyes igények között nem befolyásolható, mert ezt az algoritmus a pillanatnyi állapotok szerint intézi. Ez huzamos idejü tulterhelés esetén okozhat gondot.

Az emlitett problémák megoldása az algoritmus továbbfejlesztésével lehetséges.

6. Az algoritmus továbbfejlesztése

A számitógép rendszer sokszor olyan túlterhelési állapotba kerül, amikor a késleltetések már a külső ismérvek szerinti rendszeres késésekben jelentkeznek. Ilyenkor külső beavatkozás szükséges, amellyel a késések, vagy a késleltetések elosztását szabályozni lehet.

A szabályozás bevezetésével meg lehet határozni különböző terhelési állapotok esetére a késleltetések elosztását az egyes teljesitmény csoportok között.

A megoldáshoz be kell vezetni a *terhelési szint* (W) fogalmát, valamint ujabb paraméterekkel kell kiegésziteni a *teljesitmény-csoport* leirását.

6.1. **Definició.** Terhelési szint (W)
Az a mérték, amely megadja, hogy egy adott időintervallumban az összes kiszolgálási igény hányszorosa a felhasznált kiszolgálásnak.

$$W = \frac{\sum_i P_i(t)}{\sum_i S_i(t)}$$

ahol i az aktiv igények száma.

6.2. **Definició.** Teljesitmény csoport (Q)
Azon ismérvek összesége, amelyek meghatározzák, hogy egy felhasználói igény milyen módon juthat végrehajtása során erőforrásokhoz.

Az alap algoritmus esetén a teljesitmény-csoport $Q(t) = f(P,S(t))$ függvénnyel irható le.

A továbbfejlesztésben a függvény további változókkal bövül:

$$Q(t) = f(P,S(t), W(t),T)$$

ahol P a teljesitmény mérték, S a kiszolgálási mérték, W a terhelési szint, T a külső szabályozás szerinti periódus idő.

A Q függvény változóihoz különböző küszöbértékeket rendelve irhatjuk le az egyes teljesitmény-csoportokat.
Az i-edik csoport $Q_i(t) = f(P_i,S_i(t),W(t),T_j)$

Az információs függvény azt fejezi ki, hogy a P_i kiszolgálási igénnyel belépő feladat a T_j periódusban, $W(t)$ gépterhelési körülmények között, $S_i(t)$ kiszolgálásban részesült a t időintervallumban.

A terhelési szinttől való függés leirása

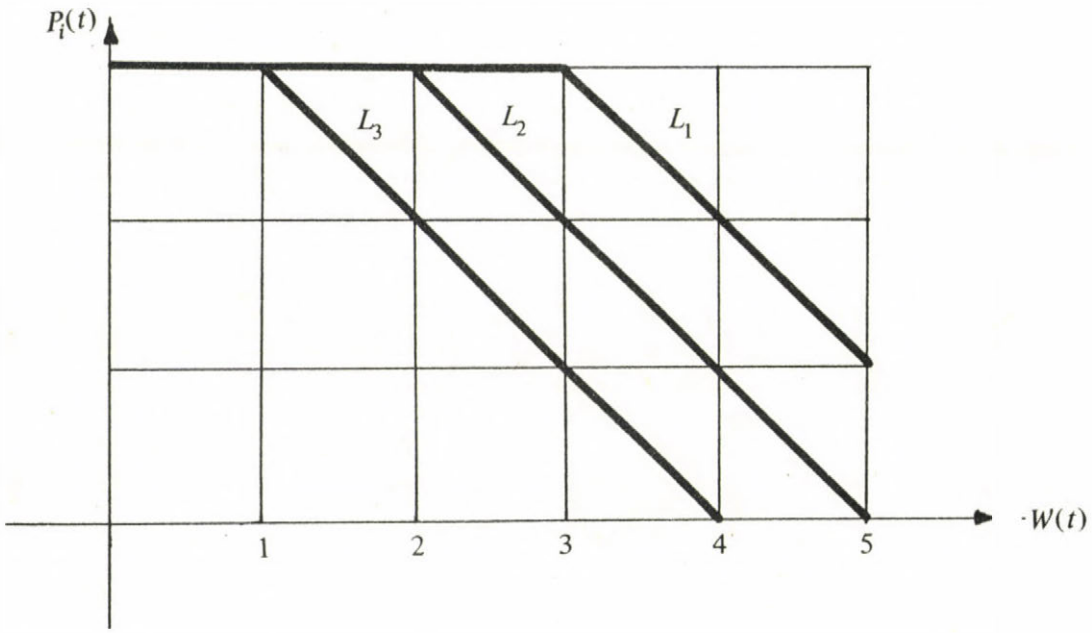
Eredeti célunk szerint azt kivánjuk elérni, hogy a különböző terhelések esetén bekövetkező késleltetéseket előre tervezett módon osszuk el az egyes teljesitmény-csoportok között.

A feladatot ugy oldhatjuk meg legegyszerübben, hogy a P_i értéket tesszük függővé a terheléstől, illetve a külső szabályozás periodusától:

$$P_i(t) = f(W(t),T_j)$$

ahol a $P_i(t_0) = P_i$, vagyis a feldolgozási igény iniciális teljesitmény mértéke.

A függvény azt fejezi ki, hogy az i-edik teljesitmény csoportban lévő feldolgozási igények a t időintervallumban $P_i(t)$ kiszolgálást igényelnek és ennek értéke a pillanatnyi terhelési szinttől és a külső szabályozási periódustól függ, kivéve a feldolgozás első, belépő esetét,amikor a függvény a P_i kezdeti, maximális értéket veszi fel.



Tehát $P_i(t)$ értékét a terhelési függvényében ilyen görbével lehet leirni. Az L segédparaméter felvételével egy görbesereget irhatunk le és ez lehetővé teszi a T_j periódussal való összekapcsolást az alábbi módon:

a Q_i teljesitmény-csoport $P_i(t)$ kiszolgálási igénye

a T_j periódusban, a $W(t)$ függvényében L_k szerint alakul.

Például:

Q_1 teljesitmény-csoport:

- p_1 kezdeti kiszolgálási igény;
- T_1 = 600 sec periódusban
- kiszolgálás L_2 szerint;
- T_2 = 60 sec periódusban
- kiszolgálás L_1 szerint;
- T_3 = futtatás befejezéséig periódusban
- kiszolgálás L_3 szerint

Példa a teljesitmény-csoportok leirására a felhasználók felé:

Q_1 = átlagos batch job-ok (2 óra átfutási idő, ha az adatok mennyisége nem haladja meg a 20 ezer tételt és a gép terhelése közepes).

Q_2 = egyéb batch job-ok (24-48 óra átfutási idővel).

Q_3 = real-time commandok (lekérdező terminálokról, 30 másodperc válaszadási idővel reggel 8-tól 11-ig).

Q_4 = real-time commandok és gyors batch jobok (CRJE terminálokról 2 perc válaszadási idővel).

Q_5 = real-time commandok (interaktiv programozási helyekről a preferált programnyelvekben 2 másodperc válaszadási idővel).

7. Összefoglalás

Az algoritmus az előre meghatározott igénnyel belépő feldolgozási feladatokat igyekszik a számitógép terhelési viszonyainak megfelelően a lehető leggyorsabban végrehajtani, amelyet a felhasznált erőforrások mennyiségének vizsgálatával ér el.

Lehetőséget ad az A, B, C rendszer paraméterek és a P_i küszöbértékek alkalmas megválasztásával az adott konfigurációhoz és feldolgozási igényekhez való illesztésre.

Biztositja a túlterheléssel müködő számitógépen bekövetkező késések megfelelő elosztásának lehetőségét a T_j és L_k paraméterek segitségével.

A müködéshez szükséges változók és paraméterek értékeinek megfelelő rögzitésével mérési adatokat bocsát rendelkezésre a rendszer üzemének vizsgálatához.

A vizsgálatok eredménye alapján az algoritmus egyszerüen módositható a már emlitett paraméterek és küszöbértékek változtatásával.

Az algoritmus gépidő terhelése az aszinkron alkalmazással minimális (a CPU várakozó állapotba kerülésekor fut, vagy amikor amugyis valamilyen ütemezési algoritmus futna). Ez egyben azt is jelenti, hogy a t időintervallum hosszusága nem tartható kézben, de erre valószinüleg nincs is szükség (bizonyitandó).

Külön problémát jelent az algoritmus beépitése egy adott számitógép és operációs rendszer környezetbe.

Az első modellezési eredmények azt mutatják, hogy az algoritmus kedvezően alkalmazható olyan számitógép környezetben ahol virtuális memória (gyors SWAP), többszörös, vagy elegendően gyors $i/0$ csatornák és a külső periféria (printer, stb.) vezérlésére szatellitek állnak rendelkezésre.

Jelenleg kisérletek folynak az algoritmus gyakorlati megvalósitására a KSH IBM 370/155-ös OS/VS1 alatt üzemelő számitógépére.

Irodalom

[1] IBM TSS/360 System reference manual (C28-2003)

[2] W. J. Doherty: Scheduling TSS/360 for responsiveness (AFIPS Proceedings FJCC, 1970)

[3] Y. Bard: Experimental evaluation of System performance IBM Systems Journal (1973/3.)

[4] IBM OS/VS1 Planning and use guide (C24-5090)

[5] IBM 370 üzemeltetési szabályok KSH.

[6] UNIVAC Real-time Verarbeitung. (Sperry Rand Univac, Zürich, 300666/500)

Summary

Dynamic resource allocation in real-time and batch environment

Peter Gyarmati

The paper examines the general properties of dynamic allocation strategies and gives the description of a new algorithm.

Р Е З Ю М Е

ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ РЕСУРСОВ В РЕАЛЬНОМ ВРЕМЕНИ И ПУЧКОВОЙ РАБОТЕ

П. Дярмати

В работе рассматриваются общие характеристики динамических стратегий распределения и дает новый алгоритм.

A konferenciát a "Számitástechnika 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.

Készült:
az ORSZÁGOS MŰSZAKI KÖNYVTÁR ÉS DOKUMENTÁCIÓS KÖZPONT
házi sokszorosítójában