An experimental automatic informational station AIST-O

by A. P. ERSHOV, G. I. KOZHUKHIN, G. P. MAKAROV, M. I. NECHERPURENKO and I. V. POTTOSIN

Computing Center of the Siberian Division of the USSR Academy of Sciences
Novosibirsk, USSR

INTRODUCTION

AIST-O is an experimental middle-scale time-sharing system. The name Automatic Informational Station (AIST) has been chosen to stress, by analogy, some new possibilities presented by time-sharing systems which give to a computation service some features of public informational and computational utility.

The development of the AIST-O station is being done in the Computing Center as a part of a more general activity known under the title "AIST project." The goal of this five-year program is (1) to provide the Computing Center by 1970 with adequate and modern centralized computing facilities which share their resources among many concurrently working users, (2) to find out what recommendations standard automatic informational stations and their software should have, and then transmit these recommendations to hardware manufactures and software designers. The "AIST project" activity has been stimulated to a great extent by the success of the time sharing development in the USA. One of the authors was acquainted with this development during his visit to this country in 1965. Some specific works in the field, namely: MAC Project CTSS,[1] Stanford Time-Sharing Project,[2] and RAND's JOSS system,[3] also influenced our approach.

The main task which has to be solved by the AIST-O construction is to obtain an initial experience in time-sharing system construction and the corresponding software development. At the same time it has to be a working system which will be in everyday use. Programming systems for AIST-O should cover to some extent all of the most interesting fields of applications of time-sharing. A special concern is to provide for the possibility of collecting various statistical characteristics of the system's functioning.

Hardware

From the engineering point of view it is not the goal to provide an optimal structure for the station. Our approach was defined, first of all, by the available equipment. Standard computers, Minsk-22, as a monitor, and two M-20 type computers, as working processors, are used. Recently a project on a multiprocessor computer system for batch processing has been elaborated in the Computing Center.[4] Many results of this project have been used in the AIST-O station for the control of the exchange with the

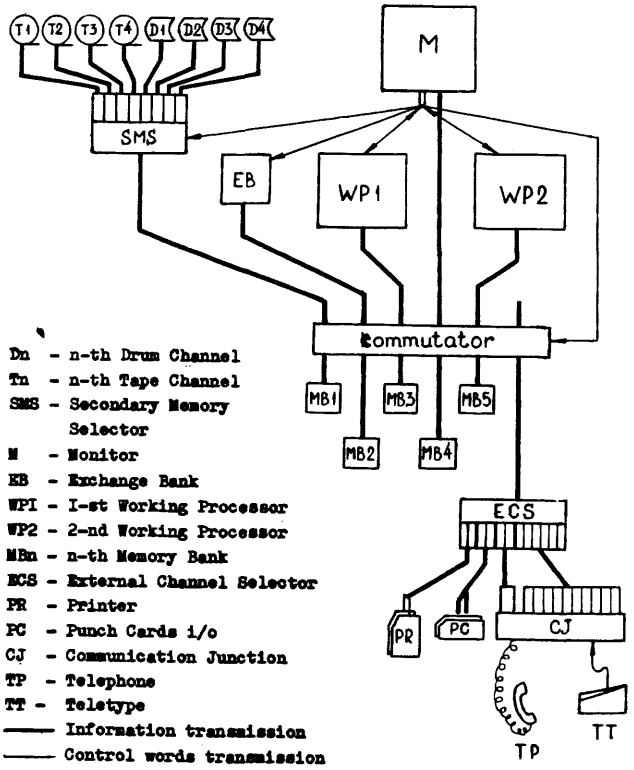

Figure 1 – The AIST-O station structure

secondary memory, for internal channel switching and external channel switching. Figure I shows the AIST-O structure.

The commutator connects through information channels the active units of the station (processors, exchange bank, secondary memory selector) with the passive ones (memory banks, and external channel selector). Working frequency of the commutator is I megacycle, the speed of channel switching is about 100 μsec. The information channels have 45 information bits, 14 address bits and some number of bits for a reserve and parity check. The information channels between the working processors and the monitor, transfer the content of the control registers when the processors are interrupted or started. The information channels between the monitor and other units (commutator, secondary memory selector, exchange bank) are used to load control words into them and to start them. An active unit having been started, operates independently and after finishing its job sends a signal of its readiness to receive the next job. The external channel selector is treated in the same manner as the memory banks are from the point of view of information transmission. The "write into the buffer" signal is also a signal to transmit the information to the corresponding channel. When the buffer has been filled an interruption of the monitor occurs, but the actual transfer of the information from the buffer is performed under the monitor control.

The external channel selector (ECS) consists of a buffer memory, multiplexing block, and a control. The buffer memory is distributed among the following external channels:

10 telegraph channels with frequency of 60 bauds,
3 telephone channels with frequency of 600 bauds,
1 card reader channel with speed of 120 45-bit words per sec.,
2 card punch or line printer channels with speed of 60 words per sec. for printing and of 12 words per sec. for punching.

Every channel is connected with the interruption system and ends with an intermediate register.

The buffer memory is distributed among the channels in the following way.

Telegraph	-	8 words
Telephone	-	32 words
Card input	-	64 words
Print and card output	-	64 words

The telegraph and telephone channels are connected through a communication junction with a standard communication network.

The ECS control consists of the buffer control, channel number decoder, serial-to-parallel and vice versa transformers, the decoder which transforms control characters (carriage return, end of text) into interruption signals and buffer counters. The ECS interrupts the monitor when transfer from a buffer zone has been finished and when half a buffer has been filled (the channel buffer works as a swing: while one half of a buffer is receiving information from the channel, the other half transfers its contents to the internal channel, then both halves change their roles).

The ECS internal channel connecting the ECS and the commutator operates as a multiplexor under the guidance of the multiplexing block. The block contains an input register with one bit per channel. The external channel bits receive a "one" during every loading of the intermediate register of the channel. The internal channel bit receives a warning "one" 6 μsec. before an information transfer. The input register bits are tested in a revert-and cyclic order, i.e., the internal channel bit is tested after every testing of the next external channel bit in turn. The speed of a test is 1 μsec. per channel, the service time is 8 μsec. per 1-word portion of information from an external channel and 8 x p μsec. per p-words portion from the internal channel. Since $p \leqslant 32$ the service time for the high priority internal channel is no more than 300 μsec. which is much less than the speed of every external channel.

The *memory banks* are standard M-20 type memory modules (4K, 6 μsec. access time). Every commutator channel has up to 4 banks, so the general information capacity of the station is expendable up to 80 K words.

The exchange bank is introduced as a fast buffer and as a means for an exchange between memory banks. The exchange bank has its own control which permits it to operate independently on the monitor after receiving from it an exchange instruction.

The secondary memory selector (SMS) permits a concurrent exchange of up to 6 memory units (tape, drum) with any memory bank or ECS buffer. The SMS has a high-speed buffer (0.8 μsec. access time), 4 drum and 4 tape channels, a control and a multiplexing block for internal channel (between the SMS and the commutator). Every external (drum or tape) channel has 3 52-bit locations in the buffer. The first location is a buffer proper, the second one contains a control word, and the third stores a current location address in the memory bank involved in the exchange.

The multiplexing block contains an 8-bit register for a cyclic input of the external channels. One input of a channel takes 1 μsec. This time is enough to transfer the buffer content either into a secondary memory unit or the input register of a memory bank. Thus, the input of all the channels takes 8 μsec. This

time is small enough in comparison with the transfer speed through any of the external channels.

The working processors are central processors of the M-20 type computers with some modifications.

- an additional register for connection with the monitor,
- possibility of internal interruptions (traps and control transfer to the monitor),
- possibility of being started by the monitor.

The monitor is a Minsk-22 type computer with the following modifications.

- additional registers for communication with memory banks and other station units,
- an interruption "system" instructions (control word transfer, starting active units, communication with interruption system registers, transfer of 45-bit words into memory banks, connection with the clock).

The clock has a timer and several interval timers, i.e., reverse time-counters which, decreasing by one with every timer signal, send an interruption signal when reaching zero.

Software

General organization. All the program support is organized by a hierarchical principle. Elements of the hierarchical structure are called "system programs." For every i-th level system program there are several (i+I)-th level system programs available to it. This means that calls for some (i + I)-th level programs make sense and can be executed by i-th level program instructions. Every system program interacts with an external channel. The system may be in one of the three modes with respect to a text coming from the channel—a text execution mode, text processing mode, and text transmission mode. Furthermore, being in one of the first two modes, a program can understand or not understand a text. A program in a transmission mode simply passes a text without any analysis to an (i + I)-th level program which is connected with the corresponding external channel. If a program in a processing mode understands a text, then it can assimilate or analyze the coming text. Some special symbols in the text can switch the program to an execution mode. If a text contains an error (i.e., it becomes non-understandable), then the program sends a message to the external channel and turns into an execution mode.

A program in an execution mode tries to understand a text as an instruction for immediate execution. If the text is clear then the corresponding instruction is executed. If the text is not understood, then an (i − I)-th level program connected with the given one is switched to the execution mode and tries to execute the text. If the text is not understood even by a first-level program (dispatcher), then a message is sent to the external channel.

If an i-th level program A in an execution mode executes a call for an (i + I)-th level program B, then the program turns into the mode of a transmission of the text to the program B which turns into an execution mode.

The total swapping time for one memory bank is 400 msec. in AIST-O. So a special care is taken to maximally reduce the net swapping time. Following are the main aids to it:

- switching working processors among various memory banks,
- overlapping a swapping in one memory bank with a working usage of another bank,
- distribution of the stream of jobs among two processors for separation of long jobs from short ones,
- floating time quantum and scheduling based on an analysis of the job stream statistics (see below).

Some psychological measures will be taken to compensate a possible lack of reactivity. First, the station will be polite in the sense that if the time to execute a job is much more than the average waiting time, then the station will immediately send to the channel a calming request to wait a little. Second, some system programs will make the station slightly talkative. More wordy comments will reduce the frequency of inquiries and, moveover, will cause some thankful feeling of comfort because owing to detailed comments of the station a user himself can answer in a more laconic and convenient form.

The dispatcher is composed of first level system programs. The dispatcher programs are executed by the monitor.

From the organizational point of view the dispatcher is considered as a collection of subroutines serving as the primary reaction for interruptions plus several subroutines controlling other station units. The primary reaction is the minimal action necessary to save control registers and to identify a system program or a service subroutine responsible for the main actions which have to be initiated by the interruption.

Here is a list of the main service subroutines:

- The interpreter of system instructions and calls for the dispatcher. System instructions are those which are sent to the dispatcher from a console when the dispatcher is in an execution mode with respect to the corresponding channel. Calls for the dispatcher are those calls for its subroutines which come from system or users' programs.

- The physical exchange organizer. This subroutine receives data coming from other dispatcher subroutines and, using these data, forms control words, sends them to the commutator and the active station units and then starts them.
- The secretary. This subroutine keeps all operative records for users on line. Its main functions are identification of users, establishing correspondence between array and program names and their physical addresses, calculating the time and other operative accounting information.
- The scheduler. This program estimates quantitative parameters of inquiries of service, organizes a line of jobs and appoints the working processors for the service. A special feature of the dispatcher is a strict separation of the process of defining or specifying inquiry parameters from the process of line formation. This makes it possible to experiment with various schedules (see below).
- The editor. On the dispatcher level there will be only minimal editing of the text (character recoding, output line formation, printing of messages sent to the consoles by the dispatcher, etc.).
- The failure control. This subroutine periodically investigates station units and responds to interruptions caused by the information transmission check.

The system programs. The hierarchic structure of the program support makes it expandable so the list given below is incomplete and lists only those programs which are under construction now.

- The bookkeeper. This program keeps accounts of all the computational and informational service given to the users. Every user is considered by the station as himself as well as a member of a certain group, which is treated by the station as one whole from some point of view. For example, the summation of the computer time is done for every user individually but the time quota is given to all the group as a whole.

  The information kept by the bookkeeper includes means of identification of users and their groups and all the accumulated information. The bookkeeper works as a part of the dispatcher and as a separate system program making some accounting operations directly for a user.
- The file maintenance program. It is difficult to establish a comfortable file system without disk-file units. The users are supposed, however, to have a tape-oriented file system which will permit them to have individual files with possibilities to store, accumulate and change alphanumerical and binary information organized in lines, words, and pages. Files will usually be identified by their names. Physical addressing will be also available but in this case a user must keep in mind some constraints imposed by the station.

  There will be intermediate buffering files on the drum to reduce the interaction time.
- The batch processor. This program will be a supervisor for batched background programs. It is appropriate to mention that there exists no "user program" notion for the dispatcher. All the jobs, both background and foreground, are executed under the supervision of some system program.
- The console symbolic debugging system. In addition to standard characteristics of such systems we would like to mention one specific feature.

  It is well known that experienced programmers or console operators can greatly help a program author without any knowledge of the essence of a problem to be solved. This is because experienced programmers have a universal strategy of searching for bugs in programs. Such a programmer, following this strategy and combining it with a specific information taken from the author's answers given to the questions put by the programmer, leads the author in a way which permits him to find out the error.

  The problem is to try to discover this universal strategy and to implement it in a system program which could be called a consultant. The consultant will come to help by a user's request and, working in a conversational mode, will lead the user to success through a sequence of debugging operations.
- The incremental ALGOL compiler. The following goals stand for the incremental compiler: to make the compilation time imperceptible to a user, to inform him immediately about any syntactical and semantic errors that appear, to formulate and put questions to a user in such a form which permits the user to correct an error or to supply the compiler with a missing information in a more laconic and compact form than required by the ALGOL syntax.

  It is supposed that the syntax analysis, statement decomposition and a partial semantic analysis will be done during the conversation. The linking and final assembling in an absolute or relocatable form will be performed when the program has been composed and put into the station. The object program can be immediately executed or transferred to an individual file.

The incremental compiler itself will be a separate and shareable program interruptable at any moment.

- The analytic manipulation system. This system program is being developed by request of the mathematical departments at the Computing Center. Executive instructions in the system are requirements for various analytical manipulations (differentiation, integration with the help of integral tables, operations over polynomials, substitutions, simplifications, parenthesis expansion and so on). It will be a universal program working in a conversational mode when the general control is in the user's hands. The system, however, will be able to accumulate the analytical instructions for their later automatic execution in an interpretative mode of operation.

Scheduling and time slicing. A multiprocessor structure of the AIST-O station offers many possibilities for experimentation in selecting scheduling algorithms. A general approach to the scheduling algorithm analysis, simulation and implementation is briefly described below.

Let us describe an environment in which the scheduler S operates (for simplicity only one working processor P is considered). In the following, a "job" is understood to be a part of a real task which requires continuous work of the processor. This means that if a job J has been interrupted (because of either exchange operation or any other reason) then the job J is considered to be finished and the necessity to continue the job is then considered as a new job J^1.

The scheduler S contains a list Q of jobs $J_1, \ldots, J_n$ standing in a line. Every job J_i has a set π_i of parameters which are necessary for the schedule composition. The processor P at the moment is running with a job J with parameters π. The job J has begun to run at the time t and has a time quantum Δ. Following are the external events for the scheduler S:

(a) Adding a new job J* to Q. A possible scheduler reaction is to interrupt job J, transform it into a job J^1 with parameters π^1. The job J* is sent to the processor P with a time quantum $\Delta^* = \Delta^*(\pi^*, Q)$. The job J^1 is added to Q.

(b) Rejection of a job from Q. A possible scheduler reaction may be an increase of Δ for the executed job J.

(c) Finishing the job J (stop or internal interruption). An obligatory scheduler reaction is sending a new job $\tilde{J}$ from Q and an appointment of a time quantum $\tilde{\Delta} = \tilde{\Delta}(\pi, Q)$ to it.

(d) Exhaustion of the time quantum Δ. An obligatory scheduler reaction is either an increase of Δ as in (b) or interrupt of J as in (a).

Let us consider in more detail a possible list of parameters of a job J. It has to be noted that some parameters characterize J not only as such but also as a part of some general task T. We consider the following characteristics to be useful.

- time t1 of entering the task T into the station,
- net processor time t2 spent to run the task T,
- number n of previous interrupts of the task T,
- supposed time t3 of completion of the task T,
- value V of the task T,
- time
- time t4 of entering job J in the list Q,
- supposed time t5 of completion of the job J,
- value C of the job J,
- time t6 for interruption and swapping of the job J.

A few words should be said of the sources and contents of these parameters. Some of the parameters (t1, t2, n, t4, t6) are directly defined by the dispatcher itself. The sources of the other parameters are system programs responsible for running the job J and the task Q. The sense of the parameters t3 and t5 is obvious and the only problem is to reliably predict them. The sense and quantification of V and C is much less clear a priori. Any inherent features of the tasks which may be important for better scheduling (for example, an absolute priority, the program length, etc.) can be related to V and C. Actually V and C can be a collection of scalar quantities.

The great degree of freedom in scheduling algorithms makes it difficult to select criteria for comparison or absolute evaluation of the algorithms. Comparison becomes more concrete if there exists a functional over a set of schedules. A degree of minimization of the functional value permits the estimation of the quality of the scheduler. Some possible alternatives for such functions are considered below.

(A) Cost functions. Let us suppose that at a moment in the list Q there appear at once n jobs $J_1, \ldots, J_n$. The solution time t_i and the cost C_i of storing J_i in the station during a time unit are known for every job. Then the total cost Φ of the solution of all the jobs is equal to

$$\Phi = \sum_{i=1}^{n} C_i T_i$$

where T_i is the time of finishing the job J_i.

It is known (for example[5] that Φ is minimized by such a schedule when all jobs are solved in turn and if they are ordered in Q by the values of the coefficient λ_i where $\lambda_i = t_i/C_i$.

This algorithm is very attractive because of its simplicity but it has some serious deficiencies. The most obvious ones are difficulties in prediction of

times t_i and in an objective choice of the job values C_i. Besides this, as it has been shown by one of the authors, appearance of a new job J_{n+1} in the list Q during previous job running can destroy the optimum reached by the previous decisions about the jobs $J_i, \ldots, J_n$. It means that to save the t/C algorithm it is necessary to be able to predict the entrance of the new jobs. Sometimes it is really possible. For example, it is possible to make some predictions about the jobs which appear after finishing the exchange operations —such jobs are simply continuations of the old interrupted tasks.

(B) Preference principles. Sometimes a scheduler tactic is formulated as a principle which sounds like something similar to "shortest programs should be served first" or "the longer a job is in the station, the higher is its priority" and so on. A typical example of such an algorithm is a well-known "Corbato algorithm."[6] If A-type algorithms may be called "economical ones" then B-type algorithms may be called "political ones." These algorithms, without raising the problem of functional minimization, guarantee a minimum station sociability with respect to users.

(C) Combination of A- and B-type algorithms. The great number of degrees of freedom and the convenient computational form of the t/C algorithm permits the organization, within the frame of a cost function, pseudo minimization, of various scheduling algorithms. A variant of the algorithm will be formed by an appropriate choice of the values C_i and an appointment of the times t_i. History accumulation can be realized by an appropriate change of C_i as a function of the time. This approach has convenient, practical considerations because by properly choosing t and C it is possible to change the algorithm's tactics with respect to a class of jobs served by the station, without changing the algorithm itself or its tactics with respect to other jobs. The possibility of adaptation of the t/C algorithm to various scheduling tactics has also been mentioned.[5]

(D) Statistical scheduling algorithms. Let us suppose that the station deals with a stationary job stream with a known net service time distribution law. Then, if for a given time t there exists a positive probability to have jobs with the net service time t, it is possible to develop a mathematical expectation F(t) of the actual service time for these jobs. Naturally, $F(t) \leqslant$ for every t. Considering F(t) and t in some interval of t it is possible to introduce various functionals characterizing the "distance" between t and F(t). These functionals should be of an integral type, and they will differ from each other, for example, in time scale (logarithmic or linear scale) and in the weight function Φ (t) which multiplies the distance between t and F(t), for example, F(t)—t. A functional having been fixed, it is possible to compare and evaluate various scheduling algorithms. One of the possible formulations of the problem is: having an a priori given function F(t) one must try to find a scheduling algorithm satisfying this function.

The authors believe that this approach may be convenient for a scheduler evaluation based on real statistics of jobs coming to the station.

REFERENCES

1 *The compatible time-sharing system: A programmer's guide*
MIT Cambridge 1965

2 *Stanford time-sharing project STSP*
Memos 1-33 Stanford University Stanford 1963 1965

3 J C SHAW
JOSS: A designer's view of an experimental on-line computing system
FJCC Proceedings 1964

4 *Some problems of multiprocessing in computing systems*
A Collection Nauka Novosibirsk in Russian 1965

5 M GREENBERGER
Priority problem
Project MAC report Cambridge 1966

6 I C PYLE
An outline of the MAC time-sharing system
STSP Memo 27 Stanford 1965