A Real-Time Monitor for Minicomputer Using Event Processing Model

Keiichi YAMAGATA* and Saburo MAKINOUCHI*

Abstract

This paper deals with a methodology for constructing control programs for real-time processing in small scale computers. A new concept *t-unit* is introduced, and the data processing is logically defined by its streams in the control function programs. The pseudo-parallel processing is performed with the modified *fork-join* of the *t-unit* streams. The *t-unit* acts as a mailbox, and it is used for synchronizing operations. The mutual linkage among the function programs is carried out dynamically at run time via the *t-unit*. The concept of *task* is not used. All the information associated with the event control and the CPU dispatching is packed in the *t-unit*; and, thus, the compact supervisor is constructed.

1. Introduction

Recently, minicomputers have been used for various kinds of control applications; and the productivity of the software systems has become an essential problem[1]. The sophisticated real-time processing is supposed to be implemented in small scale computers. Large scale computer systems have powerful real-time monitors, in which the rigid concepts of *task* (*process*), *event*, *message queue* and others have been manipulated[2]. However, these monitors are not always suitable to small scale computers because of the deficiency of memory space, large overhead and so forth. On the contrary, the compact special purpose supervisor is inadaptable to the different applications. Considering these matters, it is valuable to have the independency and the portability of the program modules; and the building block method of them is desired.

In this paper, a compact supervisor for the minicomputer NEAC-M4 (Nippon Electric Co., Ltd.) is described, which has one physical processor. It is assumed that each function program resides in the main memory. The automatic swapping is not considered here. Therefore, a swapping function is to be appended by the user if necessary.

This paper first appeared in Japanese in Joho－Shori (Journal of the Information Processing Society of Japan), Vol. 18, No. 6 (1977), pp. 526～533.

* Department of Precision Engineering, Osaka University

2. Pseudo-Parallel Data Processing

In order to perform the flexible and complex real-time processing, the new concept *t-unit* is introduced, which denotes the transition of processing requests combined with the occurrences of some events. The *t-unit* occupies 16 bytes physically in the main memory. Data processing is defined by the *t-unit* stream through a function program. The *t-unit* is an embodiment of the virtual processor; and, moreover, it has the role of a communication buffer among the function programs. The cooperation between them is executed using the information in the *t-unit*. The contents of a *t-unit* are shown in Fig. 1, which contain request codes, responses and common area. Six kinds of macro instructions are illustrated in Fig. 2. The *t-unit* streams are managed by the modified *fork-join* mechanism[3]. Even if the system has only one physical processor, a lot of *t-unit* streams may be permitted in the system; and the pseudo-parallel processing can be logically performed by this model.

In Fig. 2, by the execution of FORKT(t_a) in the stream t_m of F_a, the new stream of t_a is generated through the dispatcher. The program number which designates the function F_b is written in the *t-unit* t_a. Most macro instructions have the address of the *t-unit* as a parameter. The linkage between the function programs F_a and F_b is carried out dynamically by the transition of t_a which offers the common area to F_a and F_b. In general, the function programs are connected with each other as semicoroutines[4]. The original sender of a certain *t-unit* is always clarified; and, finally, the *t-unit* is returned to the original sender. This fact provides safety against the memory space deadlock. From the viewpoint of the user, the *t-unit* is regarded as a mailbox; and each function program could handle more than two *t-units* even if it were

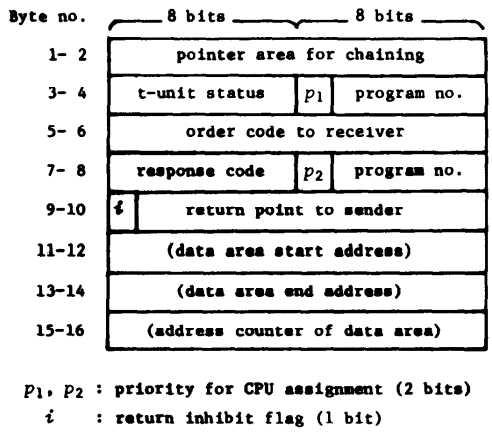

Fig. 1 The contents of a standard *t-unit*

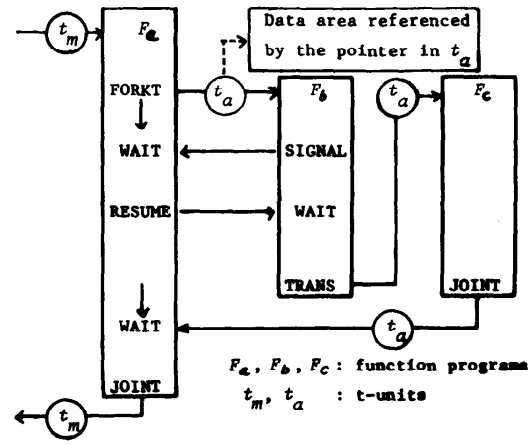

Fig. 2 Modified *fork-join* mechanism

not a reentrant type program. Using this means, it becomes feasible to manipulate the message queue and also to receive the multiple events. A function program can also deal with multiple joining points, whose addresses are indicated in each *t-unit*, respectively, as the return point (see Fig. 1).

Since the large scale monitor is not considered here, the master mode and the slave mode are not defined. Hardware interruptions are directly utilized in the user's function programs; this is called the interrupt mode. The communication between the external process and the function program is carried out through the hardware interfaces. Therefore, they are regarded as similar concepts of the *t-unit*, logically; and the interrupt analyzer acts as another dispatcher as shown in Fig. 3. In this situation, the control programs of peripheral devices can be easily replaced according to the user's requirements.

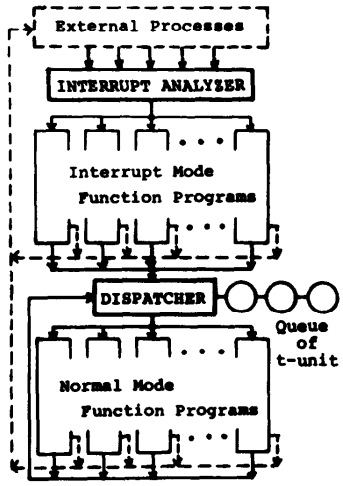

Fig. 3 A modular system structure

3. Waiting Queue in the Dispatcher

At the execution time of some macro instructions (FORKT, RESUME, TRANS), the transition requests of the *t-unit* with event information occur independently in each function program. Therefore, the distribution of the *t-unit* must be managed by the dispatcher to avoid the chaos of the overall system. For this purpose, the dispatcher has the entry point table of the function programs, in which the semaphore variable is prepared for each entry.

Most function programs are not the reentrant type, but they might deal with more than two *t-unit* streams. Handling the semaphore variables[5], it is possible to accept a number of *t-units* successively, one by one. If the acceptance condition does not hold, the *t-unit* is chained to the waiting queue in the dispatcher. In the case of JOINT or SIGNAL macro instructions, the return point and the return inhibit flag in the *t-unit* are used instead of the contents of the entry point table as seen in Fig.1. Only the basic synchronizing primitives are provided in the dispatcher; and the more complex ones, related to the application, are executed in each function program. Consequently, the queue of the event wait and that of CPU wait are arranged in one kind of queue, which is only classified by the priority of CPU assignment.

4. CPU Assignment Based on Priority Levels

The schematic diagram of the dispatcher from the viewpoint of CPU assignment is shown in Fig. 4. The waiting queue is constructed for each priority level. The highest four priority levels(0-3) are used for interrupt mode, in which the stream is activated by the hardware interfaces through the interrupt analyzer. The normal mode priority levels(5-8) are used for *t-unit* dispatching. It should be noted that each priority of the *t-unit* is indicated dynamically in the *t-unit* itself; and, thus, the priority is not fixed to the function program.

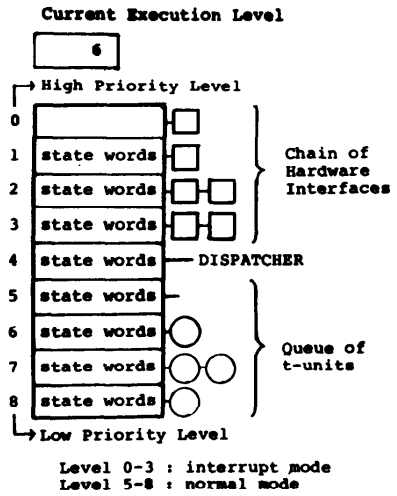

Fig. 4 Scheduling mechanism of CPU dispatching

The dispatcher always keeps the priority level of the current execution, and the level switching (CPU preemption) occurs when a higher level processing request appears at the time of interruption from the external process or at the execution time of some macro instructions. The concept of *task* is not used in this paper. The state words (the contents of the registers) are reserved by the supervisor only when the level switching occurs. However, in the case of WAIT macro instruction, only the necessary contents of the registers for the resumption must be reserved in the user's function program; because in this case, the CPU might move to the other stream without level switching. The total memory size of the state words is determined only by the number of priority levels (see Fig. 4). The number of function programs and the number of *t-unit* streams are not limited by the number of priority levels.

When more than two *t-units* are waiting in the same priority queue, they are regarded as the command sequence and processed successively so long as the conditions with the semaphores are granted (first-come first-served basis). User's function programs can manipulate both the preferential processing and the sequential processing at will with the priority handling of the *t-unit*. When one of the *t-unit* streams is selected, the entry point address or the return point address is set to the program counter; and the address of the *t-unit* is set to the base register.

Data sampling of short intervals is often required for minicomputers. In order to shorten the uninterruptable interval, the dispatcher uses one priority level(4) exclusively; and its priority is lower than that of interrupt mode as shown in Fig. 4.

Therefore, the hardware interruptions can be accepted intermittently during the run of the dispatcher. The system discussed here has 400 μsec uninterruptable time, which is independent of the queue length.

5. Concluding Remarks

By the modular approach discussed in this paper, the nucleus of the supervisor consists of the dispatcher, the interrupt analyzer and some common tables, which occupy 1.5k bytes main memory. Other executive functions and device service programs can be easily appended, since overall system generation is not necessary. The executive system for NEAC-M4 reported here is practically used for adaptive control system, which is developed to improve machining accuracy in profile milling with the equipment of KOYO-VAN-NORMAN 2GNC milling machine and the controller NEDAC-4200. Moreover, this system is connected to the TSS of the Osaka University Computation Center. The NEAC-M4 acts as the control center, and it is certified that the flexible pseudo-parallel processing by the *t-unit* stream is very effective for the adaptive control of the NC system.

The generalization of this model to multi-processor systems or multi-computer systems is very important and it is left to future research. Furthermore, high level programming language is expected, with which the pointer handling and the synchronizing operation can be easily described.

Acknowledgments

The authors are grateful to Mr. A. Kobayashi of Nippon Electric Co., Ltd. and Mr. T. Sogabe of Koyo Machine Industries Co., Ltd. for their support of the equipment.

References

1) D.L. Mills, "Executive Systems and Software Development for Minicomputers," Proc. of the IEEE, Vol. 61, No. 11, 1973, pp. 1556-1562.
2) P.B. Hansen, "Operating System Principles," Prentice-Hall, 1972.
3) M.E. Conway, "A Multiprocessor System Design," Proc. of AFIPS FJCC, Vol 24, 1963, pp. 139-146.
4) O.-J. Dahl and C.A.R. Hoare, "Hierarchical Program Structures," in Structured Programming, Academic Press, 1972, pp. 175-220.
5) E.W. Dijkstra, "The Structure of the 'THE'-Multiprogramming System," Comm. of the ACM, Vol. 11, No. 5, 1968, pp. 341-346.