Malaysian Journal of Computer Science, Vol. 9 No. 1, June 1996, pp. 12-17

REAL-TIME MULTITASKING KERNEL FOR IBM-BASED MICROCOMPUTERS

Mohammed Samaka
School of Computer Science
Universiti Sains Malaysia
11800 Penang, Malaysia
Tel.: 604-6577888
Fax: 604-6575113
email: samaka@cs.usm.my

ABSTRACT

Presents a real-time multitasking kernel (RTMK) for the IBM-based computers. It is developed using the Modula-2 Top Speed version. RTMK offers an expanded set of synchronisation and communication primitives that are dynamically created, and included with a time-out control mechanism. The kernel provides a base around which real-time application systems can be built. Its flexible and modular design simplifies the expansion of the kernel.

1.0 INTRODUCTION

In a microcomputer control system, the microprocessor must perform many functions concurrently. These include, for instance, measuring process variables, filtering signals, updating control settings, checking alarm conditions, etc. These functions require the use of computer resources such as processor, memory, and input/output devices. To perform resource sharing economically and efficiently, a number of real-time operating systems have been developed in process control applications [1, 2, 3, 4, 5]. Such operating systems can be used for building real-time, multitasking applications requiring co-ordination and resource allocation.

The aim of this study is to present a real-time multitasking kernel (RTMK) that is powerful and flexible, and allows a wide variety of application systems to be developed without much programming effort.

The RTMK provides the following features:

- Dispatching of concurrent processes competing for microprocessor use;
- Interprocess communication and synchronisation;
- A system initialisation capability;
- Handling hardware interrupts;
- Diagnostic feedback during error conditions.

2.0 IMPLEMENTATION TOOLS

The RTMK is developed on an IBM-based PC and written in a language called Modula-2, Top-Speed version. This language is largely machine independent and supports many current software engineering concepts. The Top-Speed version is facilitated with special modules that support the development of any system of congruency control [6, 7]. These modules include:-

(i) **System Module**.
It performs process creation and switching. It incorporates the following modules:

- ***New Process***. This procedure transfers execution from one process to another. The type of transfer is synchronous.
- ***Iotransfer***. Is an interrupt driven. When the processor receives an interrupt it checks whether the interrupt is associated with a process, in which case the current process is suspended, and a process that was previously suspended is resumed.

The ***System*** module also provides procedures to disable and enable hardware interrupts.

(ii) **Storage Module**.
This module allows the designer to allocate, and de-allocate blocks of memory dynamically.

3.0 PROCESS MANAGEMENT

Process is the basic active entity of an application in the RTMK. Because of the limited memory space, processes have to be dynamically created and deleted. The number of processes in the system is not fixed; it depends on the activities of the system. The principal data structure used by the RTMK to maintain a process is the process descriptor. The process descriptor contains all the information about a process such as its attributes and status. Information, such as process-name, process-state, process-priority, memory-usage, process delay/awaken time, message-address, and process-error.

It also includes the following indicators:

- ***Alarm***, a Boolean flat to indicate whether a process is using the semaphore time-out.

- ***Region***, to indicate that the process is inside the critical region.

- ***Process-wait***, to indicate the queue where the process is currently waiting.

4.0 RTMK QUEUES

The RTMK maintains a list of processes. The list usually takes the form of a queue. Processes can be added to or removed from the queue. The queue is an important data structure for process management. Within the RTMK, a number of global system queues exist. They include:

- ***Delay queue***. It holds the processes with delayed state. The queue is headed by a process with the shortest delay time.

- ***Suspend queue***. It contains the suspended process. They are ordered using the LIFO discipline.

- ***Ready queue***. It contains processes that are ready to run. Processes are ordered according to their priorities.

- ***Alarm queue***. It maintains information about time and date that a process can wait in a semaphore waiting queue.

5.0 PROCESS STATES

Each process can be in one of the four states: ***active*** (currently executed by the processor); ***ready*** (can be active if the processor is released. The other resources necessary for its continuation are available); ***waiting*** (expecting the occurrence of a certain event to continue its execution); and ***blocked*** (its existence is known to the RTMK, but is ignored by the dispatcher and may be activated by another process).

Processes that have not been created or that have been terminated are considered to be non-existent. Fig. 1 illustrates the various process states and transitions from one state to another.

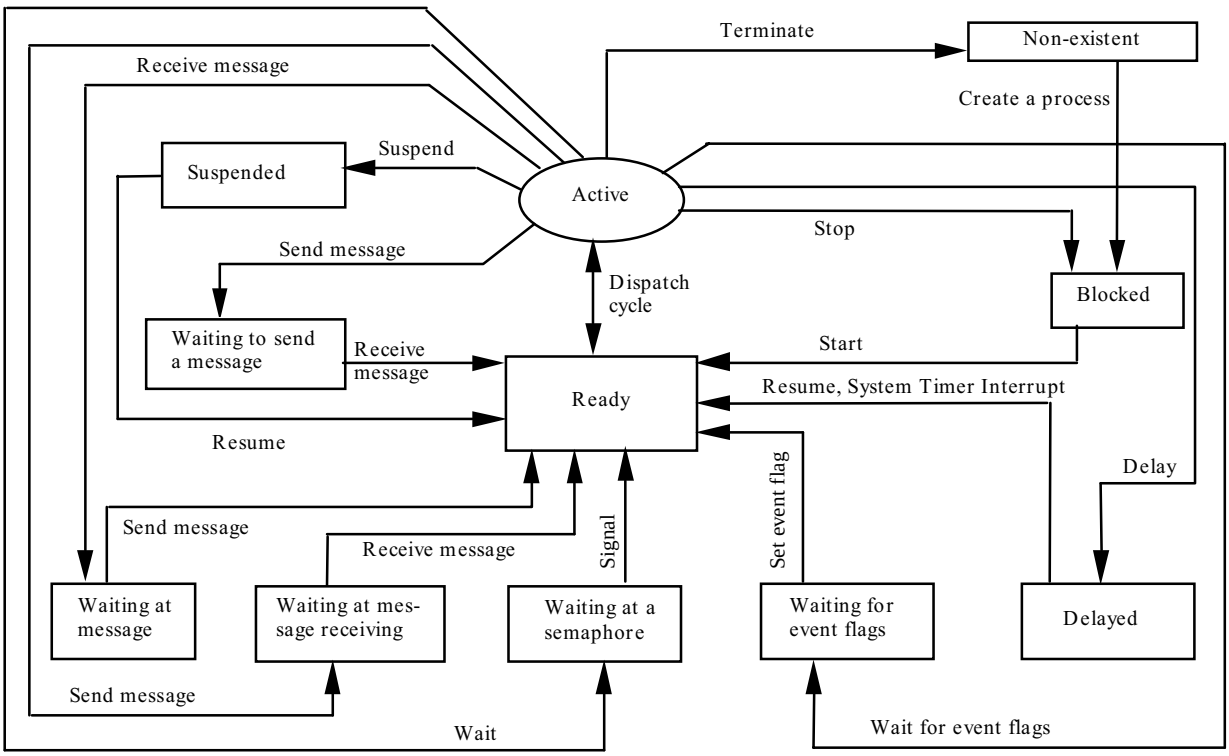

Fig. 1: RTMK process state diagram

A waiting process can expect one of the four possible conditions. When a process is waiting for a requested amount of time of elapse, it is in the delayed state. A process in the suspended state is waiting to be "resumed" by other processes.

6.0 PROCESS INTERCOMMUNICATION

In a multitasking system, processes interact for two reasons:- to share resources and to communicate with one another by exchanging information (data and messages). This leads to a number of potential problems. The most important of which are deadlocks, mutual exclusion and race condition. There must be some exchange mechanisms to cope with those problems. The mechanisms adopted in this study are: semaphores, region and mailboxes [8, 10].

6.1 Semaphores

For process synchronisation, the RTMK offers counting semaphore variables with the usual operations ***wait*** and ***send*** [8]. Semaphores are implemented using time-out mechanism. This helps to avoid the problem of deadlock. A process can wait until the resource is available or for a specified time. In this case, a process waiting time and date will be kept in a record called *alarm*. This record exists in the alarm queue. If a process waits longer than the expected time, the exception handling module will handle the problem.

Each semaphore has an integer count that indicates the number of available instances for each resource type and a queue in which it can wait. A process can wait in a semaphore waiting queue according to the specified time set by the user or it can wait until the required resource becomes available.

This implementation avoids busy-waiting, i.e. it denies CPU services to the waiting processes. When a process needs to wait on a semaphore, the system places it in a queue of processes associated with that semaphore.

6.2 Mailboxes

Co-operating processes must be able to share information. Communication between processes within the RTMK is based on message passing mechanism. It is implemented by using mailboxes, as mailboxes provide many-to-many communication. A mailbox allows multiple outstanding messages. Any process can receive a message from a mailbox. Processes are usually blocked until the request can be satisfied. The developed mailboxes are very flexible and allow another kind of communication mechanism, which is called ***rendezvous***, a synchronous bi-directional communication. A process attempting to send a message is, therefore, made to wait until a receiver is available. The receiver also awaits for a sender, so that two processes can be synchronised for a message exchange to occur [8].

Each mailbox is equipped with two semaphores and a buffer to hold messages' addresses. One semaphore controls producer, blocking any process that attempts to add messages to a full buffer. The other semaphore controls consumers, blocking any process that attempts to remove a message from an empty mailbox buffer.

The mailbox's technique provides:

- Protection against allocating large amount of memory;
- Address of messages are passed through a mailbox, not the messages themselves. Mailboxes can be used for passing any object as each object has an address. Messages can be of any data type.

6.3 Region

To ensure that only one process will be in a critical section at any one time, the RTMK provides region method, which has the same type as semaphores, but the operations are slightly different. Any process that gains access to a region cannot attempt to suspend or delete itself. Any attempt to do so will lock up the region, preventing other processes from accessing the data guarded by the region. There are two fields in a process descriptor. One is used to dedicate the nesting level of a critical section, and the other to indicate whether a process is inside the critical section.

7.0 PROCESS DELAYING/ACTIVATION

Operating systems usually use real-time clock internally to limit the amount of time a process can execute, as well as externally, to provide user programs with services like time delays. The RTMK maintains a queue of delayed processes ordered by their time. Whenever the real-time clock interrupt occurs, the delay queue will be examined and initiate the processes for which the delay time has expired. Processes in delay queue are ordered by the time at which they will be awaken. Within each process descriptor, there is a field that keeps number of ticks that the process must delay. On each clock tick, the scheduler examines processes in the delay queue and moves the process whose time delay has expired from the delayed queue to the ready queue.

8.0 INTERRUPT MANAGEMENT

A real-time system is event driven. External events occur in an unpredictable manner. Other events are caused by running processes. In many cases, these events require extremely fast response. A process may signal an event by an interrupt or system call to another process. External

events are always exsignalled by interrupts. There are two types of interrupt processing. They are:-

(i) Interrupt Process

In this type of processing, a process will be active as a result of an interrupt. Before an interrupt occurs, the process stays in a waiting state. Immediately after the interrupt occurrence, the process becomes ready, but not necessarily running because other ready processes may have greater priority in the ready queue. Interrupt process is under control of the scheduler; it has its own stack and it is used for long activities.

(ii) Interrupt Handler

For handling this type, there are two possibilities for managing events:- with or without context switching. The first uses conventional interrupt service routines, which are started when the interrupts occur. Here, the activities initiated by the interrupt are executed with a disabled interrupt and on the stack of the interrupted process. Therefore, this method is appropriate for short activities and high priority interruption. It is not under the control of the scheduler. The second is suitable for events that are signalled by interrupts. In this respect, when an interrupt occurs, immediately transfers to a procedure within a real-time system. Such procedure can establish the appropriate response and resume the interrupted process without formal rescheduling. This method is used for long activities, under the control of the scheduler.

9.0 THE SCHEDULER

Real-time processes are normally active and subject to short-term scheduling. This type of scheduling follows a strict priority algorithm with immediate pre-emption. Processes are entered into the ready queue according to their priorities. When the CPU is available, the process at the head of the ready queue is always selected. As long as there is no higher priority process in the queue, the running process continues until it requests service or voluntarily suspends itself. Because a real-time process knows best when its activities are critical, a running process may be permitted to lock out pre-emption by an appropriate system call. While this locking is active, the process will continue running even if a higher priority process becomes ready. Of course, a process is expected to use this mechanism for a short period only, and it must cancel the lock as soon as it is no longer needed.

In the RTMK, the scheduling algorithm that determines the running process is based on the process priority and state in the system. As a rule, the process selected to run is one that has the highest priority. This kind of scheduling is called pre-emptive priority based scheduling, where a higher priority process will pre-empt a lower priority running process.

Scheduling of processes with the same priority is known as ***round-robin scheduling***. In this method, each process in the ready queue is assigned a fixed CPU time. When the CPU is available, the process at the head of the queue is selected to run. If that process uses up its allocated time, it is interrupted and placed at the end of the queue. The ready processes continue to be serviced in a circular order.

When there is more than one process in the ready queue with the greatest priority, two modes are possible:- with time slicing, and without time slicing.

- ***With time slicing mode***, each process is assigned a limited time (1/18.2 sec). If the process exceeds this time quantum, a hardware interrupt triggered by the system's real-time clock will cause control to pass to a routine associated with the timer interrupt. The system state of the currently executing process is saved, and the CPU is allocated to another process in the ready queue. When a process is selected to run, the system is restored to the state before it was interrupted.

- ***Without time slicing***, the first process in the ready queue with the higher priority is executed.

10.0 EXCEPTIONAL CONDITION MANAGEMENT

It is the nature of the most real-time applications that they are left to run for many days once they are started. Therefore, RTMK must keep running despite the occurrence of software errors and hardware faults. To achieve this requirement, the exception handling mechanism is implemented. If an error occurs, the system is notified about it and an exception is generated. After an exception occurs, it is serviced by a standard supplied procedure. For this study, the exception handling mechanism is developed using assembly language 80386. Whenever a process invokes a system call, the means of communicating the success or failure of the call is the condition code. The condition codes that the RTMK return are numeric values. Conditions that represent failure are called exceptional conditions. There are two classes of exceptional conditions:- programmer errors and environmental errors. A programmer error is a condition that may be generated when calling a process. In contrast, an environmental condition is an exceptional condition that arises due to circumstances beyond the control of calling a process.

When an error occurs, the RTMK informs the user about the error by displaying a message that declares error type and the name of the process in which the error occurred, and performing a group of actions to deal with the error.

11.0 RESULTS

The most important measures used to evaluate the performance of a RTMK is the response time [9]. Response time is the time between an event (caused by an interrupt) and the reaction of the system to that event. This time depends on:-

1. How long interrupts are disabled in system operations.

2. Time of context switching.

In the present work, interrupts are disabled in operations such as ***send*** and ***wait***, to protect shared data structures (queues) against corruption. To improve the system efficiency, the time when an interrupt is disabled must be as short as possible. In general, it is not possible to measure this time without special equipment, which is not available during the period of this study.

The context switching time is the time that CPU spends in saving information about the suspended process, and restoring information about the resumed process [8]. In the present study, this time has been computed approximately for different computers, by writing a software programme. Modula-2 supplied ***SYSTEM*** module from which ***TRANSFER*** procedure can be imported. This procedure causes a direct switching from the current process to a destination process, and through this switching, information will be saved and restored about the processes. So the purpose of the programme is calculated on how many times ***TRANSFERs*** are executed during one time slice (1/18.2) Then time consumed by one ***TRANSFER*** is easily computed by the expression (1/18.2) *1000/number of ***TRANSFERS***, which means approximately the context switching time. The experiment has been applied using different computers, depending on processor type and speed, as shown in Table 1.

Table 1: RTMK performance results

COMPUTER TYPE	CLOCK FREQUENCY	CONTEXT SWITCH TIME
QUADRANT AT 80286	8 MHz	0.3969553 ms
QUADRANT AT 80286	16 MHz	0.1755433 ms
PCI NEC V 40	8 MHz	1.123276 ms
PS2 80386	20 MHz	0.1050574 ms

The context switch time plays as an important criterion in the evaluation of RTMK performance. Therefore, increase in the processes executed in the system would increase the time spent by the CPU to perform context switching. As a result, the utilisation of the system would decrease.

12.0 CONCLUSION

This research project is concerned with the design and implementation of a real-time multitasking kernel for the IBM-based computers. The kernel is written using Modula-2, Top Speed version, which implements special modules to support the development of any system of congruency control. The original goal of the RTMK formulated in the introduction has been achieved and implemented. The result of our approach is the fact that RTMK is useful for designing real-time applications for microprocessor systems.

Some essential areas of development being planned for any further future enhancements are:-

- Implementing the RTMK as an embedded controller in which the code is held in a ROM.

- The RTMK can be written as a completely independent portable product. The portability can be achieved by using an intermediate language and a virtual processor. The user object programme can be independent from the host microprocessor. That would allow the RTMK to be included in any existing microcomputer configuration due to its portability and independence.

- Another future area to look into, is the usage of ***object oriented paradigm*** in developing the RTMK.

REFERENCES

[1] K. Schevan, “Developing High-Performance Parallel Software for Real-Time Applications”, *Information and Software Technology*, Vol. 30, No. 4, May 1988.

[2] L. Nikolov and I. Kovaskki, “Design and Implementation of a Portable Kernel for Real-Time Applications”, *Microprocessing and Micro-programming*, Vol. 21, 1987, pp. 198-196.

[3] P. Gratti, “MODOSK: A Modular Distributed O.S. Kernel for Real-Time Process Control”, *Micro-processing and Microprogramming*, Vol. 9, 1982, pp. 201-214.

[4] P. Pulli, “Embedded Microcontroller Operating System with State-Machine Support”, *Microprocessing and Microprogramming*, Vol. 18, 1986, pp. 54-62.

[5] T. Bemmerl and G. Scodar, “A Portable RTMK for Embedded Microprocessor Systems”, *Micro-processing and Microprogramming*, Vol. 21, 1987, pp. 181-188.

[6] K. King, *Top Speed Modula-2 for IBM Computer*, Jenson and Partners International, 1986.

[7] B. Cornelius, *Programming with Top Speed Modula-2*, Addison Wesley, 1991.

[8] A. Tanenbaum, *Modern Operating Systems*, Prentice-Hall, 1992.

[9] T. Temelmeier, "Performance Analysis of a Micro-programmed Real-Time Operating System with an Interrupt-and-Abort Discipline", *Microprocessing and Microprogramming*, Vol. 19, 1987, pp. 233-251.

[10] W. Stalling, O. S., Prentice-Hall, 1995.

BIOGRAPHY

Mohammed Samaka obtained his M.Sc. and Ph.D of Computer Science from Loughborough University of Technology. Currently, he is a Senior Lecturer at the School of Computer Science, University Science of Malaysia. His research areas include Operating Systems, Robotics Softwares and Computer Architectures. He has published a number of papers related to these areas.